1999

Instantaneous Learning Neural Networks.

Kun Won Tang
Louisiana State University and Agricultural & Mechanical College

INSTANTANEOUS LEARNING NEURAL NETWORKS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Kun Won Tang
B.E., University of Malaya, 1978
M.S., Louisiana State University, 1997
December 1999

To my wife

Meng Kwan

and our children

Mun Ling, *Mun Yen*, and *Mun Shen*

Acknowledgments

I would like to express my deepest gratitude to Professor Subhash Kak for his invaluable guidance without which this dissertation would not have been possible. His close supervision and constant encouragement have been instrumental to the completion of this dissertation in a relatively short time.

I am also grateful to Professors Fred Denny, Jagnathan Ramanujam, Jerry Trahan, John Tyler, and Peter Wolenski for serving on my committee. The numerous constructive inputs from Professor Trahan have been particularly helpful.

Table of Contents

Acknowledgments iii

Abstract vi

Chapter

1 **Introduction** 1

2 **Instantaneous Learning Neural Networks** 10
2.1 WISARD 11
2.2 Probabilistic Neural Networks (PNN) 15
2.3 CC4 Corner Classification Neural Networks 19
2.4 Summary 25

3 **Fuzzy Classification Neural Networks** 26
3.1 Overview of an FC Network 29
3.2 The Hidden Neurons 32
3.3 The Rule Base 34
3.4 Training of the FC Network 36
3.5 Generalization by Fuzzy Membership 37
3.6 Generalization by Voting 41
3.7 FC Networks and Cover's Theorem 42
3.8 FC Networks and Curve Fitting 44
3.9 FC Networks and Kernel Regression 45
3.10 FC Networks and Radial Basis Function (RBF) Networks . 48
3.11 FC Networks and the Bayes Classifier 50
3.12 Summary 61

4 **Function Approximation and Time Series Prediction Using Fuzzy Classification Neural Networks** 64
4.1 Function Approximation 64
4.2 Time Series Prediction 68
4.3 Performance Scalability 89
4.4 Summary 92

5 **Pattern Classification Using Fuzzy Classification Neural Networks** 93
5.1 Classification by Fuzzy Memberships 97
5.2 Classification by Voting 97
5.3 Comparison Between FC and Other Networks 100
5.4 Performance Scalability 107
5.5 Summary 114

6 Conclusions 116

Bibligraphy 120

Vita 129

Abstract

Instantaneous learning is a desirable feature in neural networks. This type of learning enables the network to be trained very quickly, typically in just one or two passes of the training set as opposed to hundreds or even thousands of passes for networks trained by an iterative process, such as the error backpropagation (BP) algorithm. This dissertation first reviews several existing types of neural networks with instantaneous learning capability. It then proposes a new network called the Fuzzy Classification (FC) Neural Network that can be trained with two passes of the training samples. The first pass assigns the synaptic weights for the input and output layers. The second pass determines the radius of generalization r for each training sample. The network exhibits fuzziness in two regards: (1) by fuzzification of the location of each training vector in the input space; and (2) by assigning fuzzy memberships of output classes to new input vectors.

The operation of the FC network is analyzed from different perspectives, namely, separability of patterns, curve fitting and kernel regression. It is shown that one mode of operation of the FC network can be made to behave like a Radial Basis Function (RBF) network by proper choice of membership function and network parameter. The performance of the FC network as a pattern classifier is also discussed in a statistical framework.

The generalization performance of the FC network is compared against that of the CC4 Corner Classification Neural Network, the RBF network, and the Multilayer Perceptron network trained by the BP algorithm. Experiments involving a Henon map time series, a Mackey-Glass time series, and a spiral pattern are used in the comparison. The performance of the FC network is found to be comparable to that of RBF and BP networks, and better than that of the CC4 network. Its performance is also more scalable than those of CC4 and RBF networks. Further, the time taken to design an FC network for any given problem is much shorter compared to the other three networks.

Chapter 1

Introduction

The ability of the human brain to acquire knowledge and to make use of that knowledge to generalize has fascinated scientists and philosophers for centuries. Although serious scientific research in the past half century has resolved some of the mysteries, deeper understanding of the mechanisms by which the brain learns and generalizes has remained elusive. It was in the quest for this understanding of the inner workings of the brain and mind that the artificial neural network discipline was born. The discipline started with a small group of researchers studying networks composed of binary-valued information-processing elements, a highly abstracted version of their biological cousins at best, in an attempt to model the functioning of the brain. Today, neural network research has acquired mainstream status, although its rapid rise to the limelight has generally been attributed not so much to the interest in modeling the brain, but rather to the fact that neural networks have been successful in providing efficient solutions to problems in artificial intelligence and pattern recognition not otherwise achievable using the traditional digital computer approach.

Work on neural networks was motivated right from the beginning by the observation that the human brain processes information in an entirely different way from the conventional digital computer. The brain is a highly complex, nonlinear and massively-interconnected computer consisting of tens of billions of simple

information-processing elements called neurons. It has the capability to store information and to organize itself so as to perform certain computations such as pattern recognition, perception and motor control many times faster than the fastest digital computer ever built. For example, the brain, in conjunction with the vision system, can recognize a familiar face embedded in an unfamiliar scene within a fraction of a second. Further, its performance in such perceptual recognition tasks is often rotation- and distortion-invariant, a feat clearly well beyond any conventional computing machine today.

The speed with which the brain performs cognitive computation seems even more remarkable considering the fact that individual neurons in the cortex are slow. They are thought to have switching times of the order of several milliseconds. If the brains takes 200 milliseconds to recognize a face, the whole process of analyzing the visual data to isolate the relevant information from background scene, searching the memory database and making an identification takes at most a few hundred neuron switching cycles.

Equally impressive is the brain's ability to recall patterns seen fleetingly. This suggests the involvement of some form of instantaneous learning.

Neuroscientists have long recognized the existence of two types of memories in the human brain: long-term memory and short-term memory. Long-term memory lasts from hours to years and is responsible for the adaptive change in behavior over time. On the other hand, short-term memory, or better known as "working memory", operates over mere seconds and it appears to be the central element in the organization of behavior, language and thinking [Wickelgren, 1997; Kak, 1999]. It is supposed to

briefly store and process information for planning and reasoning [Baddeley and Sala, 1996]. The short life span of working memory suggests that it has to be acquired almost instantaneously.

Likewise, learning in artificial neural networks can loosely be classified into *instantaneous learning* and *non-instantaneous learning*. In instantaneous learning, the information to be learned is presented to the network a very small number of times, typically just once or twice, and the network remembers the information. On the other hand, in non-instantaneous learning, the neural network is presented repeatedly with the information to be learned, possibly over hundreds or even thousands of iterations. With each iteration, the network adjusts itself and gets closer to its goal of learning the information. As shall be seen later, networks trained by the iterative process are generally smaller in size than those trained by instantaneous learning. Thus there is a tradeoff between training time and network resources.

To put the work in this dissertation in perspective, some brief historical notes are in order. It is generally agreed that artificial neural network research began in 1943 when Warren McCulloch, a neurobiologist, and Walter Pitts, a statistician, proposed the first neuron model [McCulloch and Pitts, 1943]. They recognized that combining many such simple neurons into a network was the source of increased computational power. They showed that a network having a sufficient number of such neurons and with synaptic connection weights properly set can, in principle, compute any computable function. However, one limitation with their network is that it has no facility for learning or adaptation, thus the synaptic weights have to be designed into the network right from the beginning. Nevertheless, the McCulloch-Pitts neuron model

has great significance in that it represents the root of neural computing. Its historical importance is well documented in the neural network literature [e.g., Wasserman, 1989; Hecht-Nelson, 1990; Kosko, 1992; Zurada, 1992].

The first neural learning principle was postulated by D. Hebb, a psychologist at McGill University [Hebb, 1949]. His premise was based on the correlation principle: if Neuron A is stimulated repeatedly by Neuron B at times when Neuron A is active, then Neuron A will become more sensitive to stimuli from Neuron B. This postulate suggested a physical mechanism for learning at the cellular level. It led to the notion of *adjustable synaptic weights* being incorporated in most artificial neuron models as we know them today.

Some 15 years after McCulloch and Pitts proposed their artificial neuron model, Frank Rosenblatt introduced and developed a class of artificial neural networks called *perceptrons* [Rosenblatt, 1958, 1959, 1960a, 1960b, 1962]. Unlike the earlier McCulloch-Pitts networks, the perceptron can be trained to perform a particular task, such as classifying patterns. The perceptrons generated a great deal of interest in the neural network community. Their early successes resulted in enthusiastic claims [Fausett, 1994] and led some to believe that neural networks could do anything [Haykin, 1999].

Then, in 1969, Minsky and Papert pointed out some of the computational limitations of the perceptron and showed that it could not perform certain simple tasks such as deciding whether two blobs in an image are connected or not [Minsky and Papert, 1969]. The classic Exclusive-OR problem is one such task. This dampened the earlier enthusiasm on perceptrons and discouraged many researchers from continuing to

work in the field of neural networks. It brought about the "quiet years" in the history of neural networks which lasted throughout the 1970s and early 1980s.

The downfall of perceptrons was attributed to two reasons: (i) a single-layer perceptron could not solve the XOR problem (although a multilayer perceptron could); and (ii) no known methods existed at that time to train a multilayer perceptron (MLP). In general, to update the weights in a network, one must calculate an error. At the output layer, this error is easily calculated, since it is just the difference between the desired output, which is known, and the actual output observed at the output neurons. However, at the hidden layers of the MLP, it is not possible to calculate the error, since the "desired output" at these intermediate neurons are not known.

It was not until the mid-1980s that a method for training the MLP called the *backpropagation* (BP) training algorithm was brought to the knowledge of the neural network community [Parker, 1985; Rumelhart, Hinton and Williams, 1986a, 1986b; LeCun, 1986; McClelland and Rumelhart, 1988]. The theory and implementation of the BP algorithm can be found in many neural network references [e.g., Fausett, 1994; Hagan, Demuth, and Beale, 1996; Kartalopoulos, 1996; Looney, 1997; Haykin, 1999]. It is an iterative algorithm which involves presenting the training samples (input-output pairs) to the network repeatedly. Each iteration consists of two passes through the different layers of the network: a forward pass and a backward pass. During the forward pass, an input pattern is applied to the network input nodes and its effect propagates through the network layer by layer. Finally, when the effect reaches the output layer, an actual output of the network is obtained. This actual output is compared with the desired output to produce an error signal. During the backward

pass, the error signal is propagated backward through the network. It is during the backward pass that the synaptic weights are adjusted in accordance with an error-correction rule. The adjusted weight should result in the network producing an output closer to the desired output (i.e., with smaller error) if the same input pattern is presented. This procedure is repeated over many iterations and for all training samples until the network error is within an acceptable limit.

The development of the BP algorithm represented a major breakthrough in neural network research. It put to rest the pessimism about perceptron learning and generated a vigorous revival of interest in the neural network community.

Despite its success, the BP training algorithm has not escaped criticism. The issues most commonly criticized are the difficulties encountered in choosing the right network architecture for any given real-world problem, the nondeterministic nature of its convergence, and its computation-intensive nature which results in slow training speed. For example, Lin and Lee [1996] pointed out the following difficulties commonly encountered when designing a BP network (i.e., multilayer perceptron network trained by the BP algorithm):

1. Choosing the number of hidden layers that a network should have for any given real-world problem is difficult and has to be done by trial and error. Although it has been proven [Hornik, Stinchcommbe and White, 1989] that a network with one hidden layer and a sufficient number of hidden neurons can approximate virtually any function to any desired degree of accuracy, it is often essential to have two, three or even more hidden layers in the network. This is because a network with one hidden layer "... would require an impractically large number of hidden units

for many problems, whereas an adequate solution can be obtained with a tractable network size by using more than three layers (including input and output layers)." [Lin and Lee, 1996, page 242].

2. The algorithm may not converge. If it "... converges at all, it may become stuck at local minima and be unable to find satisfactory solutions." [Lin and Lee, 1996, page 243].
3. Deciding on the number of neurons in each hidden layer is difficult and has to be done experimentally. "The exact analysis of this issue is rather difficult because of the complexity of the network mapping and the nondeterministic nature of many successfully completed training procedures." [Lin and Lee, 1996, page 249].

Further, Kartalopoulos [1996, page 81] made the following observation regarding the computation-intensive nature of the BP algorithm:

> "The algorithm suffers from extensive calculations and hence, slow training speed. The time required to calculate the error derivatives and to update the weights on a given training exemplar is proportional to the size of the network. The amount of computation is proportional to the number of weights. In large networks, increasing the number of training patterns causes the learning time to increase faster than the network. The computational speed inefficiency of this algorithm has triggered effort to explore techniques that accelerate the learning time by at least a factor of two [Parker, 1987; Becker and LeCun, 1989; Cho and Kim, 1991]. Even these accelerated techniques, however, do not make the BP algorithm suitable in many real-time applications."

Looney [1997, page xvi] also made the remark that "the problems encountered in the use of backpropagation as a new tool led to some mistrust of MLPs by 1990."

From the above comments, one can imagine the frustration experienced by a designer when trying to design a BP network for solving a given problem. Indeed, when the training of a network fails, it could be due to any of the following reasons:

1. Insufficient number of hidden layers.
2. Insufficient number of neurons in the hidden layers.
3. Unsuitable initial starting weights.
4. Unsuitable learning rate.
5. Unsuitable stopping criterion.
6. The algorithm gets trapped in a local minimum.

The number of possibilities that the designer has to deal with seems prohibitively large. The designer has to adjust one or more of these parameters by trial and error before retraining the network. The slow speed of each training run further contributes to make the designer's job time-consuming.

As a result, it was deemed desirable to have neural networks with shorter training times. Several types of networks were developed as alternative to the BP network. Of particular interest to this dissertation is a class of *instantaneous learning* neural networks that can be trained in a very short amount of time and with very little computational effort compared to the BP networks. Three such networks, namely, the WISARD pattern recognition network, the Probabilistic Neural Network (PNN), and the CC4 Corner Classification Neural Network are described in the next chapter. This dissertation then proposes a new network with instantaneous learning capability called the Fuzzy Classification (FC) Neural Network.

Also of interest to this dissertation is a non-instantaneous learning neural network called the *Radial Basis Function* (RBF) Neural Network [Broomhead and Lowe, 1988; Moody and Darken, 1989]. The training of this network, though iterative in nature, is much faster than that of a BP network. The theory and implementation of

RBF networks can be found in many textbooks [e.g., Looney, 1997; Haykin, 1999] and shall not be repeated here. RBF networks, along with BP networks, are used in this dissertation for the purpose of performance comparison.

This dissertation is structured as follows. Chapter 2 takes a look at the three types of instantaneous learning neural networks mentioned above. Chapter 3 describes the new Fuzzy Classification Neural Network and analyzes its operation from different perspectives. Chapter 4 discusses the FC network as a function approximator, tests its performance using two benchmark problems in time series prediction and compares its performance to that of CC4, RBF and BP networks. Chapter 5 is similarly structured but with emphasis on pattern classification instead. Chapter 6 concludes this dissertation with discussion on advantages and disadvantages of FC networks.

All experiments in this dissertation are performed on a Pentium 166 MHz PC with 32 Mbytes of RAM running the Windows 95 Operating System. The programs are written in Matlab 5.0.

Chapter 2

Instantaneous Learning Neural Networks

A neural network has two most significant capabilities: (1) to learn from the environment in which it is embedded; and (2) to use the acquired knowledge to interpret, predict and appropriately respond to new stimuli from the external world. The first capability is called *learning* while the second is referred to as *generalization.* A neural network learns and stores knowledge about the environment through a *training* process by which network parameters such as synaptic weights and bias levels are adjusted. Some neural networks also adjust their topology during the training process.

A major task for a neural network is therefore to learn a model of the world in which it is embedded and to maintain the model sufficiently consistent with the real world so as to perform the specified task for which it is designed. Knowledge of the world is represented in the form of observations or measurements obtained by means of sensors designed to probe the environment in which the neural network is supposed to operate. These observations provide the pool of information from which *examples* used to train the neural network are drawn. The examples can be *labeled* or *unlabeled.* A labeled example consists of an input signal (i.e., training input) paired with a corresponding desired response (i.e., target output). This input-output pair is called a *training sample.* On the other hand, unlabeled examples consists of different

realizations of the input signal by itself. No desired response is provided. In any case, a set of examples, labeled or otherwise, represents knowledge about the environment of interest that a neural network can learn through training. Learning from labeled examples is referred to as *supervised learning* while *unsupervised learning* refers to the use of unlabeled examples. Supervised learning trains a neural network to *map* input patterns to some output classes while unsupervised learning trains it to *cluster* input patterns in some meaningful fashion.

This chapter takes a look at three existing supervised learning neural networks with instantaneous learning capability, namely, the WISARD pattern recognition network, the Probabilistic Neural Network (PNN), and the CC4 Corner Classification neural network.

2.1 WISARD

WISARD (**WI**lkie, **S**tonham, **A**leksander **R**ecognition **D**evice) [Aleksander, Thomas and Bowden, 1984; Aleksander, 1989; Aleksander and Morton, 1990, 1991, 1993] is a pattern recognition network with instantaneous learning capability. It does not use weighted connections between neurons. Instead, neuron functions are implemented as lookup tables stored in commercially available Random Access Memory (RAM) chips. Rather than adjusting weights, training in a WISARD network consists of changing the contents of the lookup table in the RAM. Since RAM is a binary logic device, input and output data in a WISARD network must be digitized into binary vectors.

Figure 1 shows a basic RAM node which forms the building block of a WISARD network [Aleksander and Stonham, 1979]. A RAM node consists of the following:

1. A set of N input lines which constitutes a binary input vector.
2. An output line that returns a binary value.
3. A memory (RAM) device which is accessed by being given an "address" represented by the input vector.
4. A value-in line for programming the content of the RAM.
5. A learn/recall control line to select the operation to be carried out. "Learn" puts the RAM node in write mode during training while "recall" puts it in read mode during testing.

The RAM node operates by performing a logical function and returning a value for a given input vector. The input signals at the address lines constitute an N-bit binary vector that can address 2^N individual memory locations. Each location is one bit wide and stores either a 0 or a 1. Thus a single RAM node can perform mapping from 2^N distinct N-bit input vectors to their respective 1-bit outputs. Training of a RAM node consists of writing data into the corresponding locations addressed by the training input vectors. Learning is thus instantaneous.

Figure 2 shows k units of N-input RAM nodes arranged in a layer to form a *discriminator*. Network input is kN bits long. Connections between network input and the RAM node input lines are random, with each bit in the network input mapped randomly to one input line in a RAM node. At the discriminator output, an adder sums the outputs of the RAM nodes to produce the discriminator's response r. Before

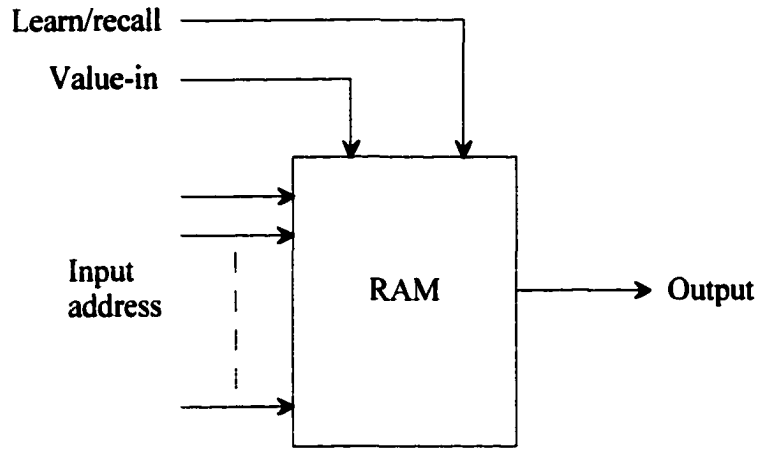

Fig. 1: A RAM node

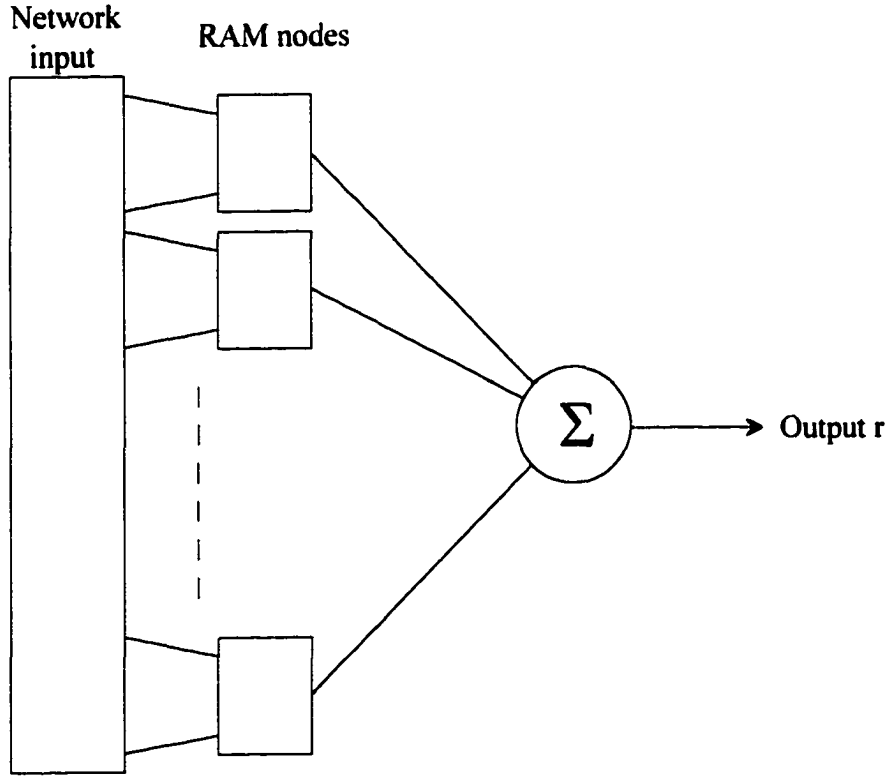

Fig. 2: A discriminator

training, all $k2^N$ locations in the discriminator are set to 0. Training an input pattern **v** is done by presenting **v** at the discriminator's input, setting the RAM nodes into write mode, and writing a 1 into all locations addressed by the input pattern **v**. If **v** is presented to the discriminator again at a later time with the RAM node set to read mode, such previously written locations will all be accessed and every RAM node will respond with 1. This leads the discriminator response to be maximum, i.e., $r = k$. If a noisy version **u** of **v** is presented, r would be a function of the number of previously written locations that are accessed by **u**, which is proportional to the similarity between **v** and **u**. This is the mechanism by which the network generalizes.

A WISARD network consists of a set of such discriminators connected in parallel, with one discriminator trained to recognize a different class of patterns. Network input is presented to all discriminators simultaneously. The WISARD network assigns an unknown test pattern **u** to the class corresponding to the discriminator with the highest response r.

It has been reported in the literature [Aleksander and Morton, 1993] that the WISARD network is particularly good at recognizing the expression on a person's face. In one such experiment, a WISARD network used only two discriminators, one trained for frowns and the other for smiles. Each input image was digitized into 128-by-128 binary points. It was reported that, after being trained on less than ten faces, the WISARD network was able to recognize a smile or a frown on a previously unseen face.

Despite its excellent performance in pattern recognition, the WISARD network has a serious limitation in that it is not suitable for use in applications involving function approximation, such as time series prediction. The fact that input and output data must be digitized into binary vectors also constitutes an inconvenience for its users.

2.2 Probabilistic Neural Networks (PNN)

The PNN [Specht, 1988, 1990, 1996] is another pattern classification neural network with instantaneous learning capability. Its principle of operation is based on statistical technique that combines the Bayes strategy for decision making with a nonparametric estimator for probability density function. The particular estimator used in the PNN is of the form

$$f_k(\mathbf{x}) = [(2\pi)^{R/2}\, \sigma^R\, S]^{-1} \sum_{(i=1..S)} \exp[-(\mathbf{x} - \mathbf{x}_{ki})^T(\mathbf{x} - \mathbf{x}_{ki}) / 2\sigma^2] \qquad (2.1)$$

where

k = category

$f_k(\mathbf{x})$ = likelihood of pattern $\mathbf{x}$ belonging to category k

i = sample number

S = total number of training samples

$\mathbf{x}$ = input vector

$\mathbf{x}_{ki}$ = ith training sample from category k

σ = smoothing parameter

R = dimensionality of input space

This estimator attempts to estimate from the training set, the underlying probability density function as a sum of small multivariate Gaussian distributions centered at each training sample.

A PNN network for a three-class problem is shown in Figure 3. Network inputs are *R*-dimensional continuous-valued vectors normalized to unit length. The *input units* receive these input vectors and feed them to all the *pattern units*. Each pattern unit forms a dot product of the input vector $\mathbf{x}$ with a weight vector $\mathbf{w}_i$ (i.e., $z_i = \mathbf{x} \cdot \mathbf{w}_i$) and then performs a nonlinear operation on z_i before outputting its activation level to the *summation unit*. One summation unit is required for each output category while one pattern unit is needed for each training sample. The nonlinear activation function used in the pattern unit is an exponential function of the form $\exp[(z_i - 1)/\sigma^2]$ as shown in Figure 4. The smoothing parameter, σ, is determined experimentally. A range of values for σ between 0.2 and 0.3 is reported to give good performance [Specht, 1996].

The network is trained by setting the $\mathbf{w}_i$ weight vector in each of the pattern units equal to one of the $\mathbf{x}$ vectors in the training set and then connecting the pattern unit's output to the appropriate summation unit with a connection weight of 1. Thus, training of the network is accomplished with just one presentation of each training vector.

Since both the $\mathbf{x}$ and $\mathbf{w}_i$ vectors have been normalized to unit length, the expression

$$\exp[(z_i - 1)/\sigma^2]$$

is equivalent to

$$\exp[-(\mathbf{w}_i - \mathbf{x})^T(\mathbf{w}_i - \mathbf{x})/2\sigma^2]$$

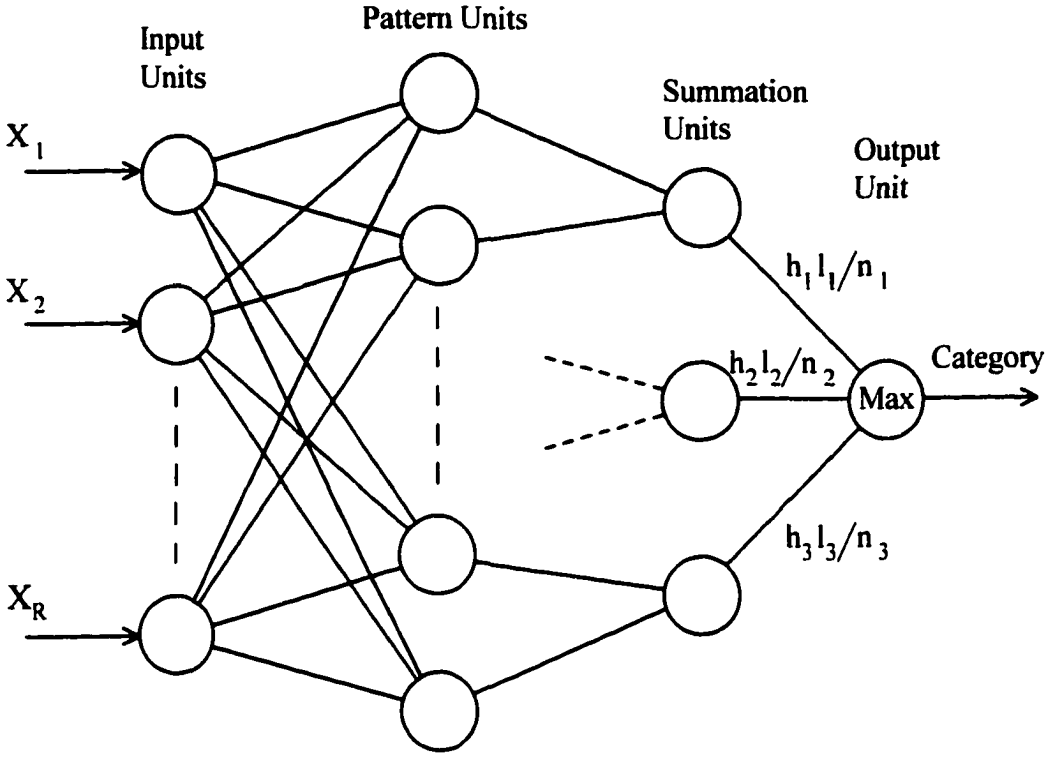

Fig. 3: A PNN with 3 output classes

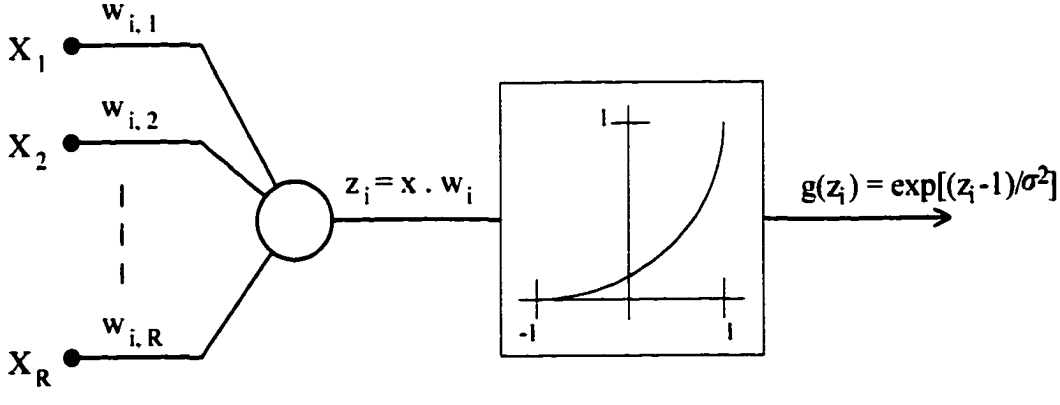

Fig. 4: A PNN pattern unit

which is the same form as Eq. (2.1). The summation units simply sum the output of the pattern units that correspond to the category from which the training patterns were selected. Thus, the PNN is effectively a direct embodiment of the probability density function estimator described by Eq. (2.1).

From Bayes decision theory for a multi-class problem, it is well known that a test pattern should be assigned to category k if

$$h_k l_k f_k(\mathbf{x}) > h_q l_q f_q(\mathbf{x}) \quad \text{for all } q \neq k$$

where

h_k = a priori probability of occurrence of patterns from category k

l_k = loss associated with classifying a test pattern into category other than k when, in reality, it belongs to category k

Applying this decision theory, the output from each summation unit is first multiplied by $h_k l_k / n_k$ before being fed to the *output unit*. The quantity n_k is the number of patterns in the training set that belong to category k. The output unit then performs the function of a *maximum detector* and determines the category into which the test vector should be classified.

While the theory behind the PNN is easily understood, it is not clear from the references cited how, in a multi-class problem, the losses l_k can be determined reliably for each category in a real-world problem. It is likely that these losses have to be determined heuristically or by a trial-and-error process. Determination of the a priori probabilities h_k for each category may also be problematic when the training set is small.

The PNN suffers the same limitation as the WISARD network in that it is not suitable for applications involving function approximation. For such applications, a variation of the PNN, called the *General Regression Neural Network* (GRNN), has been proposed [Specht, 1991, 1996]. It uses the regression technique in conjunction with a nonparametric estimator to estimate the underlying probability density function $f(\mathbf{x}, y)$ from the training set $(\mathbf{x}_i, y_i)$, $i = 1, 2, \ldots S$. It is similar to the PNN in network architecture and thus also possesses the instantaneous learning capability.

2.3 CC4 Corner Classification Neural Networks

The corner classification (CC) family of neural networks was first proposed in 1992 [Kak, 1992] and subsequently granted a U.S. patent in 1995 [Kak and Pastor, 1995]. Three versions, namely CC1, CC2 and CC3, were originally proposed [Kak, 1993; 1994; 1995]. Training in CC1 network is iterative in nature. CC2 is the first network in the family to incorporate instantaneous learning. However, it only performs input/output mapping, without any generalization capability. CC3 overcomes the problem of CC2 by introducing the concept of *radius of generalization* which enables it to generalize. By this concept, any test vector whose Hamming distance from a training vector is smaller than the radius of generalization of the network is classified in the same output class as that training vector.

In 1997, a fourth member, the CC4 network, was added to the family [Tang, 1997; Tang and Kak, 1997; Tang and Kak, 1998; Kak, 1998]. A CC4 network differs from earlier CC networks in two regards: (1) it uses one hidden neuron to learn one training sample; and (2) it allows *inhibitory* weights in the output layer. It has been

shown to work significantly better than earlier CC networks [Tang, 1997]. It has also been successfully incorporated into a stock trading system [Mehta, 1997] and an Internet metasearch engine [Shu and Kak, 1999] with very impressive results. This section describes the CC4 network in particular, with emphasis from the instantaneous learning perspective.

The CC4 network uses a three-layer feedforward architecture as shown in Figure 5. The input and output layers are fully connected. The number of input nodes in the network is equal to the total length of the input vector. An additional node, which has a constant input of 1, is added to the input layer to serve as the bias for the hidden neurons. The number of hidden neurons is equal to the number of training samples the network is required to learn, with each hidden neuron being used to learn one sample specifically. The number of output neurons is equal to the length of the output vector.

The same activation function, namely, the binary step function, is used in both the hidden and output neurons. This function, as shown in Figure 6, outputs a 1 if the sum of all weighted inputs is greater than zero; and outputs a 0 otherwise.

Before input and output data can be used to train the network, they must be digitized and encoded into binary vectors using the *unary encoding* scheme. Toward this end, a suitable *quantization level*, q, which is the number of bits used to digitized the data, must be selected. In the unary encoding scheme, the number of bits allocated for quantization determines the largest number that can be digitized. The number of 1's in a unary-encoded number equals the value of the number itself. Thus, if $q = 10$, then the bit pattern (0000000001) represents the number 1, (0000000011) represents the

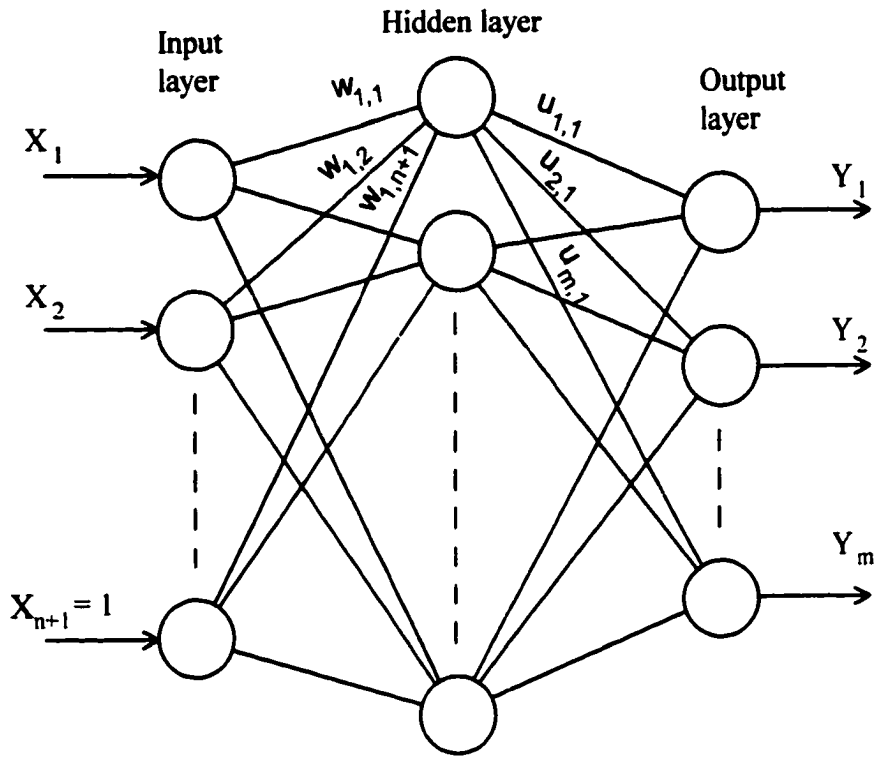

Fig. 5: General CC4 network architecture

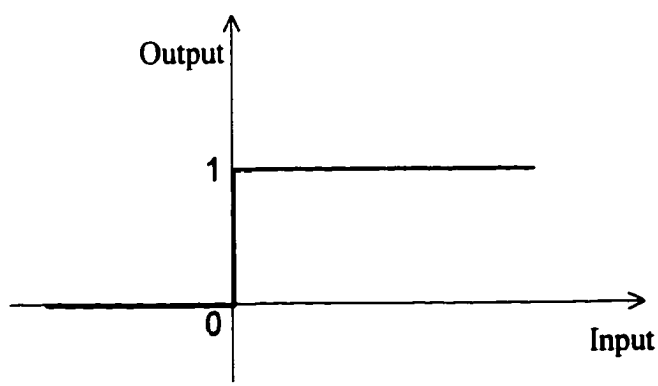

Fig. 6: Binary step function

number 2, etc., and the largest number that can be digitized is 10, encoded as (1111111111). The value of q should therefore be chosen such that it is big enough for the network to handle the largest input ever to be encountered. If the network input comprises more than one variable, each variable must be digitized using a suitable q. The resultant network input is a binary vector obtained by concatenating the bit patterns of the respective variables.

Another network parameter that must be chosen before training can commence is the *radius of generalization*, r. This parameter directly affects the performance of the network after training. It is a function of the number of training samples to be learned and the total length of the concatenated input vector. Its optimum value is determined heuristically.

To train the network, input and output weights are prescribed simply by inspection of the training input-output pairs. For each training vector presented to the network, if an input node receives a 1, its weight to the hidden neuron corresponding to this training vector is set to 1. Otherwise, it is set to -1. The bias node is treated differently. If s is the number of 1's in the training vector and the radius of generalization of the network is r, then the weight between the bias node and the hidden neuron corresponding to this training vector is $r - s + 1$. Similarly, at the output layer, if the target output contains a 1 at an output neuron, the weight from its hidden neuron to that output neuron is set to 1, which represents an excitatory weight. Otherwise, the weight is set to -1, which has an inhibitory effect on the output neuron. This training algorithm can be formally stated as follows:

```
for each training input vector x_i
    s_i = no. of 1's in x_i;
    for j = 1 to n                    // n = length of input vector x_i
        if x_ij = 1
            w_ij = 1;                 // Input weight
        else
            w_ij = -1;
        end
    end
    w_i,n+1 = r - s_i + 1;            // r = radius of generalization

    for k = 1 to m                    // m = length of target output vector y_i
        if y_i,k = 1
            u_k,i = 1;                // Output weight, excitatory
        else
            u_k,i = -1;               // Output weight, inhibitory
        end
    end
end
```

Thus, it can be seen that learning in a CC4 network is instantaneous, with each training sample presented to the network only once to be learned. The information regarding the input-output mapping that the network is required to perform is stored explicitly in the form of network weights.

The reason for prescribing weights in the above manner can be understood more easily by first assuming r to be 0. When a training input vector is presented to the network, the hidden neuron corresponding to this training sample receives a 1 input from those input nodes which are presented with a 1 and zero input from all other input nodes which are presented with a 0. At the same time, it receives a contribution of $-s + 1$ from the bias node. Thus the total input received by this hidden neuron is $s - s + 1 = 1$. The binary step activation function therefore causes this hidden neuron to fire. All other hidden neurons receive zero or net negative input because of the mismatch in positions of the +1 inputs and the +1 weights. Thus only one hidden neuron will fire

for each training vector presented to the network. At the output layer, the way in which the output weights have been assigned ensures that the single firing hidden neuron produces the correct output. The input-output mapping is therefore accomplished. The hidden neuron corresponding to a training vector is said to be *positively correlated* to the training vector while all other hidden neurons are said to be *negatively correlated* to it.

If a test vector which is not in the training set is now presented to the network, all hidden neurons will receive zero or net negative input because of the negative correlation and no output neuron will fire. This test vector is said to be *blocked* by the network. Thus it is easy to see that when $r = 0$, the network will learn all training samples correctly but will not generalize at all.

Consider now $r > 0$. When a test vector is presented, the network will block it unless it is sufficiently similar to any of the training vectors which the network has already learned and stored. The condition that the test vector be "sufficiently similar to a stored vector" requires that its Hamming distance from that stored vector be less than the radius of generalization of the network. If the test vector is sufficiently similar to more than one training vector, the output of the network is determined by the hidden neuron that corresponds to the largest network output. This is the basic mechanism by which the CC4 network performs generalization.

As with the WISARD network, the requirement that input and output data be digitized into binary vectors represents a disadvantage in the CC4 network. However, unlike the WISARD and the PNN, the CC4 network can be used for both pattern classification and function approximation applications.

2.4 Summary

It is clear that the three types of networks described in this chapter differ greatly in their principles of operation. The WISARD relies on a lookup table implemented in RAM chips for pattern matching; the PNN has its principle of operation deeply rooted in statistical techniques; while the CC4 network uses Hamming distance between binary vectors for classification. However, they all share one common characteristic: the use of one hidden neuron (or discriminator in the case of WISARD) to learn each training sample. In so doing, these networks store the training samples in a way that the samples are mutually independent of one another. When the network updates itself to learn a new sample, those samples that have been learned previously are not affected in any way.

By contrast, in conventional neural networks trained by an iterative process, the training samples are not stored in a mutually independent manner. When the network updates its weights in response to a new training input, the behavior of the network to earlier samples also changes. As a result, all samples must be presented repeatedly over many iterations.

It shall be seen that the new network to be proposed in the next chapter also relies on this mutual independence between stored vectors to achieve its instantaneous learning capability.

Chapter 3

Fuzzy Classification Neural Networks

The human brain is an extremely complicated organ by any measure. Early researchers recognized that it was difficult to describe the brain's total behavior in its full complexity. They built upon experimental results in cognitive psychology to identify basic functionality underlying a certain behavior. They attempted to isolate mental faculties in the brain and to describe them in explicit mathematical terms. One such effort is the famous book, *Laws of Thought*, by Boole in 1854. Boole constructed a mathematical system, now called *Boolean algebra*, that was explicitly motivated by cognitive psychology [Valiant, 1994].

Boolean algebra formally resembles traditional algebra but it is intended to operate in a binary domain. Thus, a Boolean variable x can take only values "true" or "false" rather than numerical values such as 2 or -10.8 as in traditional algebra. As a result, the meaning of some basic operations also need to be different, since conventional arithmetic operations, such as multiplication, no longer make sense in the binary domain. The purpose of a Boolean variable is to represent the truth value of a proposition. Thus x can stand for "it is raining" and y for "it is cold." A Boolean operation can then be used to create new propositions from old ones. For example, the operation AND can be used to create the proposition "x AND y" which in this instance would stand for the proposition "it is raining and it is cold." Likewise, the operation

OR is used to create the proposition "x OR y" which would stand for the proposition "it is raining or it is cold or possibly both." A third operation is NOT which is used to represent the negation of a proposition. Thus, "NOT x" would denote that "it is not raining." Boolean algebra is concerned with the laws under which expressions formed by Boolean operations can be manipulated. For example, "(NOT x) AND (NOT y)" is deemed to be equivalent to "NOT (x OR y)", since the two expressions produce the same truth values for all possible combinations of truth values of x and y.

An important aspect of Boole's contribution is that it provides a model of cognition in which variables can take on a discrete choice of values, namely "true" or "false", rather than an unlimited choice of values. However, the world we live in is *graded*, and most cognitive tasks performed by the brain involve some degree of *fuzziness*. For example, upon touching two objects, we perceive them as "cold" by virtue of their giving us a sensation sufficiently similar to one given be a particular cold object that we have touched before. However, one object may be colder than the other. Thus, the proposition "it is cold" would possess a higher truth value for one object than for the other. It is this premise of fuzziness upon which the neural network being proposed in this chapter is based, thus it is named the *Fuzzy Classification* (FC) Neural Network.

Although its name contains the word "fuzzy", the FC network is distinctly different from fuzzy neural systems commonly encountered. Existing systems can broadly be classified into three categories [Lin and Lee, 1996; Jang, Sun, and Mitzutani, 1997]:

1. Neural fuzzy systems: These are fuzzy logic systems with automatic tuning capability typical of neural networks. Their basic functionality such as fuzzification of inputs, fuzzy logic base, inference engine, and defuzzification of outputs remain unchanged. Neural networks are used to augment numerical processing of fuzzy sets, such as to elicit membership function.
2. Fuzzy neural systems: These are simply fuzzification of conventional neural networks. They retain the basic architecture and properties of the original neural networks. Fuzzification is typically effected by replacing the crisp neurons in the network with *fuzzy neurons* and by using a fuzzy relation in place of the standard crisp activation function.
3. Fuzzy-neural hybrid systems: These systems consist of standard neural networks and fuzzy logic systems working together as subsystems and in synergistic integration, each serving its respective function in the hybrid system. The subsystems complement each other to achieve a common goal.

By contrast, fuzziness in the FC network, as shall be seen later, arises from the way it assigns partial memberships of output classes to a test pattern. It does not use any fuzzy neurons in the network.

The principle of operation of the FC network relies heavily on the nearest-neighbor (NN) classification concept which includes the single-nearest-neighbor (1NN) rule and its extension, the *k*-nearest-neighbor (*k*NN) rule. Classification by the NN technique is not new. Indeed it was introduced in the early 1950s [Fix and Hodges, 1951; 1952]. Since their conception, the NN rules have attracted many followers. They were widely researched [Cover and Hart, 1967; Cover, 1968; Fukunaga, 1972;

Duda and Hart, 1973; Devijver and Kittler, 1982; Watanabe, 1985] and improved upon over the years [Hart, 1968; Hellman, 1970; Tomek, 1976; Dasarathy, 1977; Devijver, 1979].

With the introduction of fuzzy set theory by Zadeh in 1965, researchers have found numerous ways of incorporating fuzzy concepts into the NN rules in an attempt to get better performance. Bezdek [1981] suggested that interesting and useful algorithms could result from the allocation of fuzzy class membership to the input vector. Jozwik [1983] proposed an iterative method for finding the optimum values of k and the elements in the fuzzy membership array in order to minimize the error rate. Keller et al. [1985] incorporated fuzzy set methods into conventional crisp kNN algorithm. Bezdek [1986] defined a formalism for NN rules and proposed a framework that unified fuzzy and crisp kNN algorithms. The work in this dissertation incorporates the 1NN and kNN classifiers into a neural network framework. Perhaps more important is that the network possesses the desirable instantaneous learning capability.

3.1 Overview of an FC Network

Figure 7 shows the configuration of a general FC network. It uses the fully-connected feedforward network architecture consisting of a layer of input nodes, a layer of hidden neurons, a rule base followed by an output layer. The number of output neurons is determined by the problem specifications. For simplicity, only one output neuron has been shown, although the network can be easily extended to multiple outputs. In fact, a network with multiple output neurons can always be separated into multiple single-output networks.

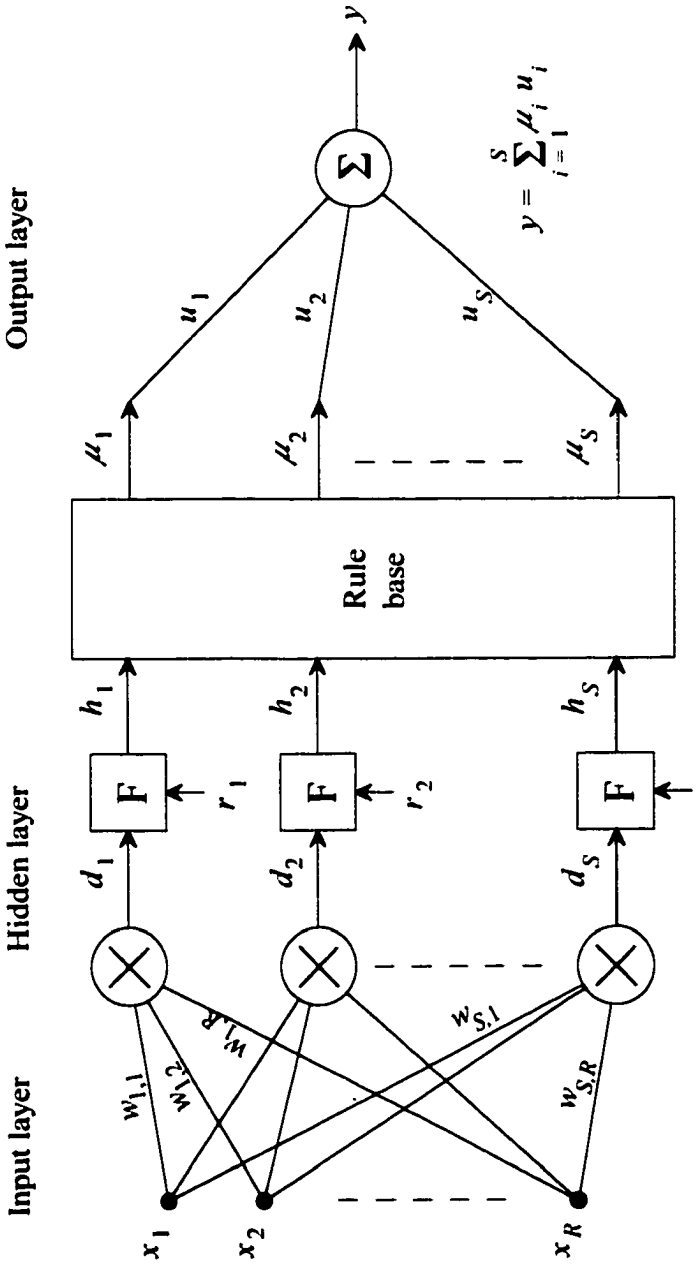

Fig. 7: FC network architecture

Input data are normalized to the range [0, 1] and presented to the network in the form of an R-element long continuous-valued vector $\mathbf{x} = (x_1, x_2, ..., x_R)$, where R is determined by the problem specifications. Like the CC4 network, the number of hidden neurons S in an FC network is equal to the number of training samples the network is required to learn. In other words, one hidden neuron is charged with learning one training sample. Thus, the hidden neurons in an FC network are mutually independent, a necessary condition for instantaneous learning.

Each hidden neuron i ($i = 1, 2, ..., S$) is associated with a weight vector $\mathbf{w}_i = (w_{i,1}, w_{i,2}, ..., w_{i,R})$. The convention used in assigning indices to these weights is as follows. The first index indicates the particular hidden neuron destination for that weight. The second index indicates the source of the signal fed to that hidden neuron. Thus, $w_{i,j}$ is the weight *from* source x_j in the input vector *to* the ith hidden neuron. For the output layer, since there is only one destination neuron, the destination index is dropped.

Each hidden neuron i first computes the normalized Euclidean distance d_i between its weight vector $\mathbf{w}_i$ and the test vector $\mathbf{x}$. This distance together with r_i, the *radius of generalization* for this hidden neuron, constitute the inputs to the activation function $\mathbf{F}$ which determines the hidden neuron output h_i. The output of all hidden neurons taken together forms the distance vector $\mathbf{h} = (h_1, h_2, ..., h_S)$ that gives a measure of similarity between the test vector and *each* of the training vectors the network has already learned.

The rule base enables the FC network to generalize. It consists of a set of IF-THEN rules that operates on the distance vector $\mathbf{h}$ to produce a *fuzzy* membership

grade vector $\mu = (\mu_1, \mu_2, ..., \mu_S)$ that indicates to what degree the test vector **x** belongs to each of the network output classes. The output neuron then computes the dot product between the output weight vector $\mathbf{u} = (u_1, u_2, ..., u_S)$ and the membership vector μ to produce the generalized network output y corresponding to test vector **x**.

Training of an FC network involves two distinct steps. The first step determines the input and output weights while the second step finds the radius of generalization for each hidden neuron. It shall be seen that each of these steps requires just one presentation of the training data set. This gives the network its instantaneous learning capability.

3.2 The Hidden Neurons

An FC hidden neuron is shown in Figure 8. Unlike the neurons in a CC4 network, the input vector **x**, the weight vector **w**, as well as the scalar output h in the FC neuron are all continuous-valued quantities. Quantity d is the Euclidean distance between **x** and **w**. The parameter r is called the radius of generalization, a term borrowed from the CC4 network. As shall be seen later, its purpose is to allocate to this hidden neuron, an *area of generalization* in the input space with radius r and its center at **w**. Unlike the CC4 network where the same r applies to the whole network, the radius of generalization in an FC network is individually determined for each hidden neuron during training. Further, the training process used in an FC network ensures that there is no overlapping between the area of generalization for each training sample.

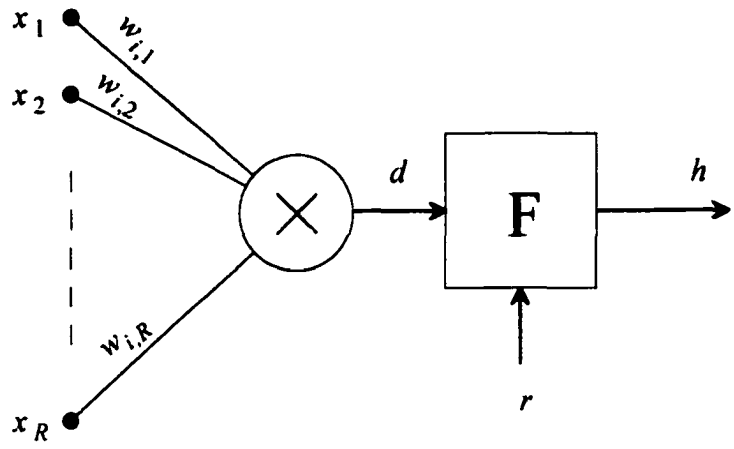

Fig. 8: FC hidden neuron

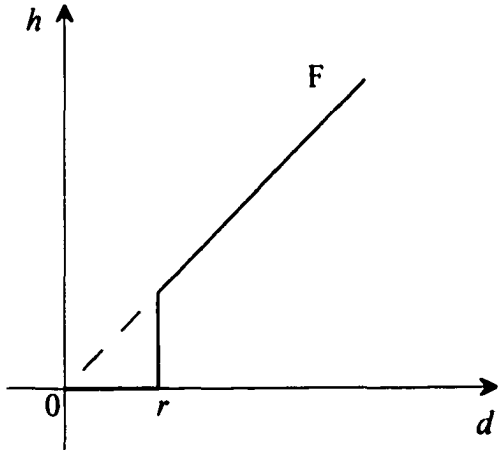

Fig. 9: Activation function F

The shape of the activation function F is shown in Figure 9. It takes two scalar inputs, r and d, and produces a scalar output h for the hidden neuron. Its operation can be described mathematically as

$$\begin{aligned} h &= 0 \qquad d \leqslant r \\ h &= d \qquad d > r \end{aligned} \tag{3.1}$$

The effect of this activation function is to replace any distance between **x** and **w** less than or equal to r by zero. Equivalently, it may be viewed as a *fuzzification* of the location occupied by **w** in the input space from a *crisp* point with zero radius into a *fuzzy* area with radius r. From this point of view, any test vector **x** within a distance r from **w** will be indistinguishable from **w**. If **w**, the weight vector for this hidden neuron, is made identical to a training vector **v**, then **x** will also be indistinguishable from **v**, and will therefore be classified into the same output class as **v**. As shall be seen later, the training process does make the weight vector for each hidden neuron equal to its training vector. The process of fuzzification of the location of **w** is one of two mechanisms by which an FC network performs generalization.

3.3 The Rule Base

Another mechanism by which an FC network performs generalization is to treat a test vector **x** as having *fuzzy* membership grades in output classes of its nearest neighbors. The function of the rule base is to ascertain these membership grades, i.e., the degree to which **x** belongs to these output classes. In an FC network, as in a CC4 network, each training vector is mapped to an output. However, in a CC4 network, a test vector either belongs to or does not belong to an output class. In other words, membership grade in

a CC4 network is *crisp*, taking on only 0 or 1 as its possible values. If the distance between the test vector and a training sample is less than the radius of generalization of the network, its membership grade in that output class is 1. Otherwise, its membership grade in that output class is 0. By contrast, membership grade in an FC network is *fuzzy*. It can take on any value between 0 and 1 depending on the distance from the test vector to the training sample. Value 0 indicates non-membership while 1 indicates full membership.

Only the k training samples nearest to the test vector in the Euclidean sense are considered when assigning membership grades. The value of k is typically a small fraction of the size of the training set. Membership grades are normalized, i.e., the sum of all membership grades equals 1. By comparison, in a CC4 network, membership grades are not always normalized. If the test vector falls within the radius of generalization of two or more training samples, it takes on membership grade of 1 in each of their output classes. Its total membership grade thus adds up to more than 1. Similarly, if it does not fall within the radius of generalization of any training sample, its total membership grade equals zero.

The rule base consists of two IF-THEN rules that assign fuzzy membership grades μ_i based on the outputs of the hidden neurons. Let m be the number of hidden neuron outputs h_i that equal 0. As shall be seen later, the training process ensures that the area of generalization for each training sample does not overlap with that for another sample. A test vector can therefore fall within the area of generalization of at most one training sample. This means that m can only be 0 or 1. The two IF-THEN rules within the rule base are as follows:

Rule 1: IF $m = 1$, THEN assign μ_i using single-nearest-neighbor (1NN) heuristic

Rule 2: IF $m = 0$, THEN assign μ_i using k-nearest-neighbor (kNN) heuristic

Note that, to avoid confusion, the 1NN and kNN rules are referred to as heuristics. The operation of the rule base shall be discussed in Section 3.5 below.

3.4 Training of the FC Network

Training the FC network involves two separate steps. In the first step, input and output weights are *prescribed* simply by inspection of the training input/output pairs. For the ith training input ($i = 1, 2, ..., S$) presented to the network, input weight $w_{i,j}$ is made equal to x_j ($j = 1, 2, ..., R$) while the output weight u_i is made equal to the corresponding target output y_i; i.e.,

$$w_{i,j} = x_j, \quad u_i = y_i, \quad (i = 1, 2, ..., S;\ j = 1, 2, ..., R)$$

In the second step of the training process, the radius of generalization for each hidden neuron is determined. This is accomplished by another presentation of the training set. When the ith training input is presented, d_i will be zero, since the distance from training vector i to itself is zero. The smallest non-zero distance, d_{min}, is thus the distance to its nearest neighbor, say training vector j. The radius of generalization for the ith hidden neuron, r_i, is then set to $d_{min}/2$. This will ensure that the area of generalization of hidden neuron i will not overlap with another hidden neuron. These two steps complete the training of the FC network. Learning in an FC network is therefore instantaneous, with each training sample presented to the network only twice.

Note that although the method used to determine r ensures no overlapping, it does not guarantee complete coverage of the input space. This is because the nearest

neighbor of training vector j is not necessarily training vector i, in which case r_j will be *smaller* than r_i. However, this is of no concern, as Rule 2 in the rule base is designed to handle this situation.

3.5 Generalization by Fuzzy Membership

With the completion of the training process, the FC network is ready to be deployed. When a test vector **x** is presented to the network, the outputs of the hidden neurons form the distance vector $\mathbf{h} = (h_1, h_2, ..., h_S)$ that gives a measure of similarity between the test vector and each of the training vectors the network has already learned. The rule base operates on this distance vector and produces a set of membership grades $\mu = (\mu_1, \mu_2, ..., \mu_S)$ according to the 1NN heuristic or the kNN heuristic. The output neuron then computes the dot product between the output weight vector $\mathbf{u} = (u_1, u_2, ..., u_S)$ and the membership grade vector to produce the generalized network output y corresponding to test vector **x**. The network output y can thus be written as

$$y = \sum_{i=1..S} \mu_i u_i \tag{3.2}$$

The operation of the 1NN heuristic and the kNN heuristic shall be described below.

3.5.1 1NN Heuristic

The 1NN heuristic is used when exactly one element in the distance vector **h** is 0, i.e., when the test vector falls within the area of generalization of a training sample. It therefore belongs to the output class of that training sample with a membership grade of 1 and to all other classes with a membership grade of 0. Thus, if h_j is zero, the membership grades are assigned as follows:

$$\mu_i = 1 \text{ if } i = j \quad (i = 1, 2, ..., S)$$

$\mu_i = 0$ if $i \neq j$

3.5.2 *k*NN Heuristic

The *k*NN heuristic is used when none of the elements in the distance vector **h** is equal to 0, i.e., when the test vector **x** does not fall within the area of generalization of *any* training sample. The test vector is assigned fuzzy memberships of classes whose training samples are its k nearest neighbors. These training samples correspond to the k hidden neurons with the smallest outputs. Membership grades for all other classes whose training samples are not in the set of k nearest neighbors of **x** are set to zero. The discussion below begins with $k = 2$ and then generalizes k to higher values.

Figure 10 shows the point X, representing a test vector **x**, and two points A and B, representing two training samples, **a** and **b**, which are the nearest neighbors of **x**. Let the distances from **x** to **a** and **b** be a and b respectively. The triangular membership functions $\mu_{\text{close to a}}$ and $\mu_{\text{close to b}}$ characterize the change in membership grades of **x** as its distances from **a** and **b** change. For example, if $a = 0$, then **x** falls within the area of generalization of **a**. Its membership grade in **a**'s class is 1 and that in **b**'s class is 0. The *k*NN heuristic then reduces to the 1NN heuristic. A similar situation exists if $b = 0$. In between these two extreme cases, **x**'s membership grades in these two classes, denoted by $\mu(\mathbf{x}, \mathbf{a})$ and $\mu(\mathbf{x}, \mathbf{b})$ respectively, change linearly as

$$\mu(\mathbf{x}, \mathbf{a}) = b/(a + b)$$
$$\mu(\mathbf{x}, \mathbf{b}) = a/(a + b) \tag{3.3}$$

This is the *k*NN heuristic for $k = 2$. Note that, for clarity in presentation, the notation μ_i used earlier has been changed to the form $\mu(\mathbf{x}, \mathbf{a})$.

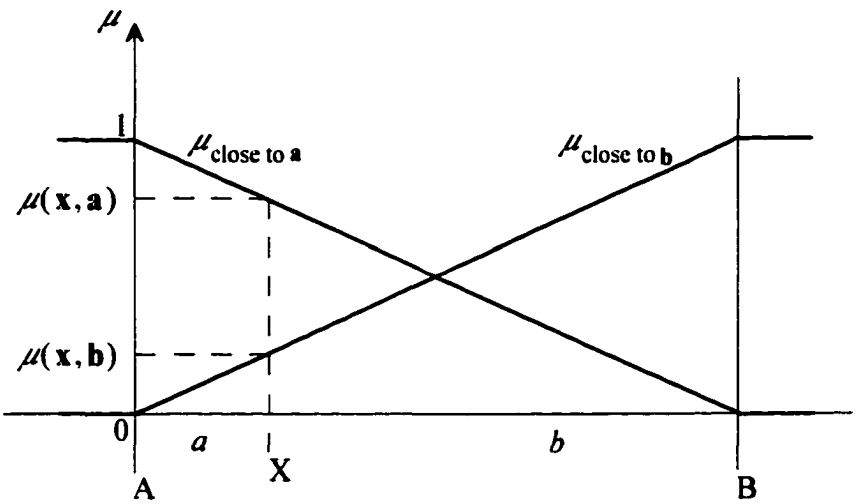

Fig. 10: Triangular membership function

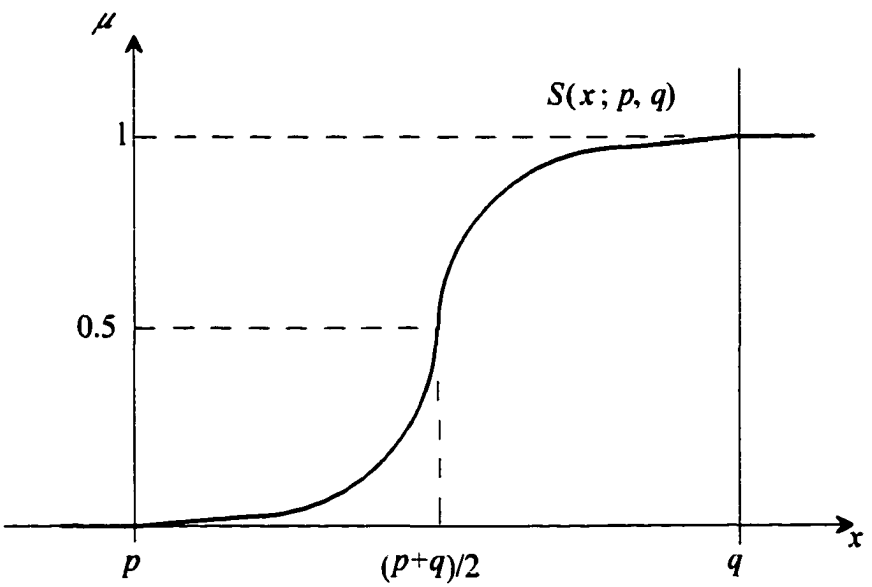

Fig. 11: *S* membership function

The above results can be generalized to higher values of k. Eq. 3.3 can be rewritten as

$$\mu(\mathbf{x}, \mathbf{a}) = (1/a)/(1/a + 1/b)$$
$$\mu(\mathbf{x}, \mathbf{b}) = (1/b)/(1/a + 1/b) \tag{3.4}$$

It is easy to see that, for example, for $k = 3$ and $\mathbf{c}$ as the third nearest neighbor at a distance c from $\mathbf{x}$, the respective membership grades are

$$\mu(\mathbf{x}, \mathbf{a}) = (1/a)/(1/a + 1/b + 1/c)$$
$$\mu(\mathbf{x}, \mathbf{b}) = (1/b)/(1/a + 1/b + 1/c) \tag{3.5}$$
$$\mu(\mathbf{x}, \mathbf{c}) = (1/c)/(1/a + 1/b + 1/c)$$

Results for higher values of k can be derived in a similar fashion. It is easy to verify that the membership grades sum up to 1 in each case.

For simplicity, the above discussion is based on the triangular membership function. Other membership functions are of course possible. For example, Figure 11 shows a quadratic function known as the *S function* in fuzzy set literature [Lin and Lee, 1996]. It is defined mathematically as:

$$\mu(x; p, q) = \begin{cases} 0 & \text{for } x < p \\ 2\{(x - p)/(q - p)\}^2 & \text{for } p \le x < (p + q)/2 \\ 1 - 2\{(x - q)/(q - p)\}^2 & \text{for } (p + q)/2 \le x < q \\ 1 & \text{for } x \ge q \end{cases} \tag{3.6}$$

The membership grades $\mu(\mathbf{x}, \mathbf{a})$, $\mu(\mathbf{x}, \mathbf{b})$, etc., would vary with the squares of the respective distances when this quadratic membership function is used.

Similarly, other distance metrics such as the city block distance could be used in place of the Euclidean distance metric. Preliminary testing indicates that the performance of the network is not seriously affected by the choice of distance metric and membership function.

3.6 Generalization by Voting

The previous method of generalization can be viewed as one in which each of the k nearest neighbors casts a *fuzzy* vote whose weight is inversely proportional to its distance from the test vector. In this section, the notion of fuzziness shall be removed and each vote shall carry the same weight, regardless of the distance from the test vector. The output class that gets the largest number of the votes wins. The performance of these two modes of operation of the FC network shall be tested in Chapter 5. To distinguish between the two methods, the former shall be called *classification by fuzzy membership* while the second method shall be referred to as *classification by voting.*

In a general problem where there are more than two output classes, the voting process may end in a tie in a boundary region where three or more classes meet. In such a case, the tie can be broken by any arbitrary means that is convenient to implement, such as picking one of the tied classes randomly. For a problem with just two output classes, a tie can be avoided simply by choosing k to be odd.

The modification from classification by fuzzy membership to classification by voting can be effected by changing membership grades such as $\mu(\mathbf{x}, \mathbf{a})$, $\mu(\mathbf{x}, \mathbf{b})$, etc. in Section 3.5.2 from fuzzy to crisp. In other words, if a training sample belongs to the set

of k nearest neighbors of the test vector, its component in the membership vector μ is set to 1. Otherwise, its component is set to 0.

The operation of the FC network shall now be analyzed from different perspectives.

3.7 FC Networks and Cover's Theorem

In the context of a pattern classifier, the FC network can be viewed as performing a nonlinear transformation from the input space to the *hidden space*, followed by a linear transformation to the output space. The hidden space is invariably of much higher dimensionality than the input space; indeed, its dimensionality is equal to the number of samples to be learned. The mathematical justification for such a nonlinear transformation to a high-dimensional space followed by a linear transformation can be traced back to *Cover's theorem* on the *separability of patterns*, which may be stated qualitatively as follows [Cover, 1965]:

> A complex pattern-classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space.

Once the patterns become linearly separable, the classification problem can be readily solved by the output neurons.

As an illustration, consider the classic Exclusive-OR (XOR) problem. The input patterns (vectors) in the two-dimensional input space are (0, 0), (0, 1), (1, 0), and (1, 1). The corresponding outputs are 0, 1, 1, and 0 respectively. These patterns are not linearly separable in the input space, as Figure 12(a) shows. The FC network hidden layer transforms these patterns into four-dimensional vectors (1, 0, 0, 0), (0, 1, 0, 0),

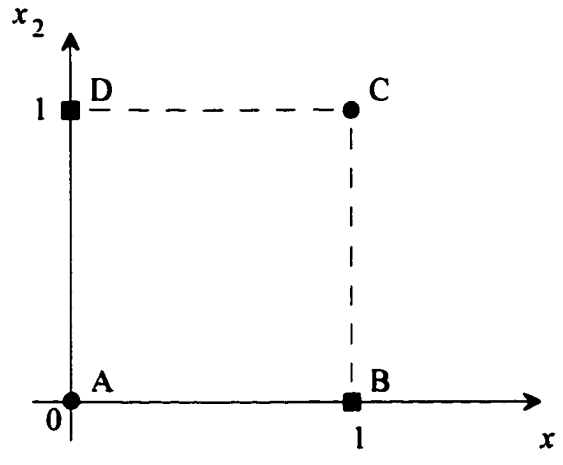

Fig. 12(a): Exclusive-OR (XOR) problem

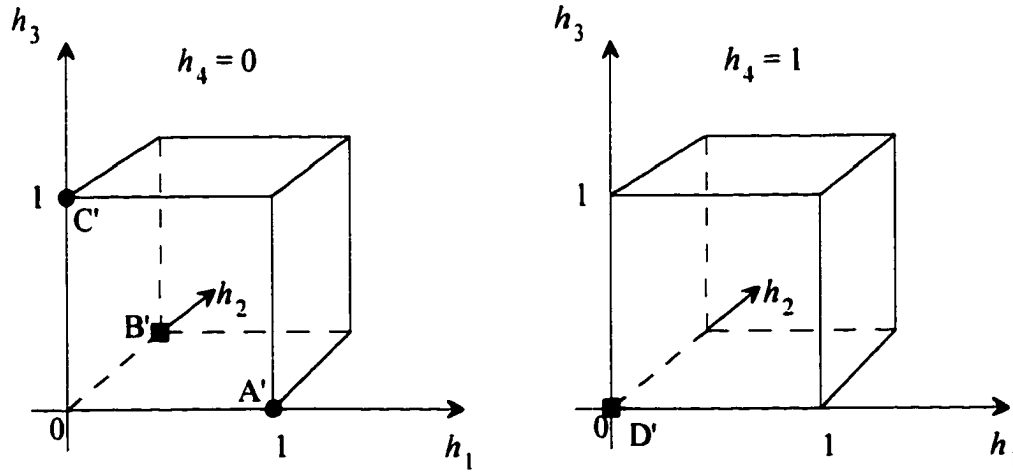

Fig. 12(b): XOR vectors in hidden space

(0, 0, 1, 0), and (0, 0, 0, 1) respectively in the hidden space. These vectors occupy the corners of a four-dimensional hypercube, as shown in Figure 12(b), and they become linearly separable.

3.8 FC Networks and Curve Fitting

In the context of function approximation, neural network *learning* can be viewed as *curve fitting* in a multidimensional input space while *generalization* is equivalent to using this multidimensional surface to *interpolate* the test data. In its *strict* sense, the interpolation problem may be stated as follows [Haykin, 1999]:

> Given a set of S different points $\{\mathbf{x}_i \in \mathbf{R}^R \mid i = 1, 2, ..., S\}$ and a corresponding set of S numbers $\{y_i \in \mathbf{R}^1 \mid i = 1, 2, ..., S\}$, find a function $F : \mathbf{R}^R \rightarrow \mathbf{R}^1$ that satisfies the interpolation condition:
>
> $$F(\mathbf{x}_i) = y_i, \qquad i = 1, 2, ..., S$$

In other words, the function F is constrained to pass through *all* the training data points. A BP network performs the curve fitting by attempting to find a surface that provides a *best fit* to the training data. Thus, the function implemented by a BP network may not pass through all the data points. By contrast, the FC network always provides an *exact* fit to the training data, since presentation of any vector in the training set would invoke the 1NN heuristic, which in turn guarantees that the network delivers the correct target output. Thus, the FC network always fulfills the above interpolation condition. The Radial Basis Function (RBF) network is also capable of fitting the training data exactly provided that one hidden neuron is used to learn one sample.

The proficiency of each network in using the fitted surface to interpolate test data, i.e., generalization, shall be compared in the next chapter.

3.9 FC Networks and Kernel Regression

The principle of operation of the FC network presented earlier has been based on the notion of fuzzy memberships of output classes. In this section, another viewpoint based on a nonparametric regression technique commonly used in statistics, namely, *kernel regression* [Wand and Jones, 1995; Ryan, 1997; Haykin, 1999] shall be taken. This approach is closely related to the notion of *density estimation* [Rosenblatt, 1956; Whittle, 1958; Parzen, 1962] which in turn is similar to the problem of estimating the spectral density function of a stationary time series [Parzen, 1961].

Consider a nonlinear regression problem over the data set $(\mathbf{x}_i, y_i)$ described by the model

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, 2, ..., S \tag{3.7}$$

where ε_i is a sample drawn from a white noise process of zero mean and $f(\mathbf{x})$ is assumed to be smooth, in the sense that similar inputs correspond to similar outputs. The problem is to estimate the underlying unknown regression function of the model, $f(\mathbf{x})$, given the training data set $(\mathbf{x}_i, y_i)$.

A reasonable estimate of the function in the vicinity of a point $\mathbf{x}$ would be the mean of the values of y observed around $\mathbf{x}$, or more precisely, the conditional mean of y given $\mathbf{x}$ (also known as the *regression of y on* $\mathbf{x}$). Thus, $f(\mathbf{x})$ may be written as:

$$f(\mathbf{x}) = E[y \mid \mathbf{x}] \tag{3.8}$$

Using the formula for the expectation of a random variable, this can be rewritten as

$$f(\mathbf{x}) = \int_{-\infty}^{\infty} y f_Y(y \mid \mathbf{x}) dy \tag{3.9}$$

where $f_Y(y \mid \mathbf{x})$ is the conditional probability density function (pdf) of Y, given $\mathbf{x}$. Now, from probability theory,

$$f_Y(y \mid \mathbf{x}) = f_{\mathbf{X},Y}(\mathbf{x}, y)/f_{\mathbf{X}}(\mathbf{x}) \tag{3.10}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is the pdf of $\mathbf{x}$ and $f_{\mathbf{X},Y}(\mathbf{x}, y)$ is the joint pdf of $\mathbf{x}$ and y. Thus, Eq. (3.9) becomes

$$f(\mathbf{x}) = \int_{-\infty}^{\infty} y f_{\mathbf{X},Y}(\mathbf{x}, y)dy / f_{\mathbf{X}}(\mathbf{x}) \tag{3.11}$$

The problem of estimating $f(\mathbf{x})$ has thus been transformed into one of estimating $f_{\mathbf{X},Y}(\mathbf{x}, y)$ and $f_{\mathbf{X}}(\mathbf{x})$ from the training data set $(\mathbf{x}_i, y_i)$, $i = 1, 2, ..., S$.

Toward this end, a nonparametric estimator known as the *Parzen-Rosenblatt density estimator* [Rosenblatt, 1956, 1970; Parzen, 1962; Cacoullos, 1966] may be used. The formulation of this estimator is based on a *kernel*, denoted by $K(\mathbf{x})$, which has properties similar to those associated with a probability density function:

- The kernel $K(\mathbf{x})$ is a continuous. bounded, and real function of $\mathbf{x}$, and symmetric about the origin where it attains its maximum value.
- The total volume under the surface of the kernel $K(\mathbf{x})$ is unity; i.e., for an R-dimensional vector $\mathbf{x}$,

$$\int K(\mathbf{x})\, d\mathbf{x} = 1 \text{ where the integral is taken over } \mathbf{R}^R.$$

Assuming that $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_S$ are independent random vectors and identically distributed, the Parzen-Rosenblatt density estimate of $f_{\mathbf{X}}(\mathbf{x})$ is defined as follows:

$$f'_{\mathbf{X}}(\mathbf{x}) = (1/Sh^R)\sum_{(i=1,S)} K((\mathbf{x} - \mathbf{x}_i)/h) \text{ for } \mathbf{x} \in \mathbf{R}^R \tag{3.12}$$

where the smoothing parameter h is a positive number that controls the size of the kernel.

Parzen [1962] proved that this density estimator asymptotically approaches the underlying probability density function provided that it is smooth and continuous. Thus, if $h = h(S)$ is chosen as a function of S such that

$$\lim_{(S \to \infty)} h(S) = 0,$$

then

$$\lim_{(S \to \infty)} E[f'_{\mathbf{X}}(\mathbf{x})] = f_{\mathbf{X}}(\mathbf{x})$$

Similarly, the Parzen-Rosenblatt density estimate of the joint probability density function $f_{\mathbf{X},Y}(\mathbf{x}, y)$ may be formulated as follows:

$$f'_{\mathbf{X},Y}(\mathbf{x}, y) = (1/Sh^{R+1})\sum_{(i=1,S)} K((\mathbf{x} - \mathbf{x}_i)/h)K((y - y_i)/h) \quad \text{for } \mathbf{x} \in \mathbf{R}^R \text{ and } y \in \mathbf{R} \qquad (3.13)$$

Thus,

$$\int_{-\infty}^{\infty} y f'_{\mathbf{X},Y}(\mathbf{x}, y)dy = (1/Sh^{R+1})\sum_{(i=1,S)} K((\mathbf{x} - \mathbf{x}_i)/h) \int_{-\infty}^{\infty} yK((y - y_i)/h)dy \qquad (3.14)$$

Changing the variable of integration by letting $z = (y - y_i)/h$, and using the symmetric property of the kernel $K(\bullet)$, Eq. (3.14) becomes

$$\int_{-\infty}^{\infty} y f'_{\mathbf{X},Y}(\mathbf{x}, y)dy = (1/Sh^{R})\sum_{(i=1,S)} y_i K((\mathbf{x} - \mathbf{x}_i)/h) \qquad (3.15)$$

Eq. (3.15) and Eq. (3.12) therefore provide an estimate for the terms in the numerator and denominator of Eq. (3.11) respectively. Performing the substitution followed by some simplification, Eq. (3.11) becomes

$$F(\mathbf{x}) = f'(\mathbf{x}) = \sum_{(i=1,S)} y_i K((\mathbf{x} - \mathbf{x}_i)/h) \,/\, \sum_{(j=1,S)} K((\mathbf{x} - \mathbf{x}_j)/h) \qquad (3.16)$$

where, for clarity of presentation, the index of summation in the denominator has been changed to j instead.

The relationship between the FC network and kernel regression can be seen by defining a *weighting function* as

$$W_{S,i}(\mathbf{x}) = K((\mathbf{x} - \mathbf{x}_i)/h) \,/\, \Sigma_{(j=1,S)} K((\mathbf{x} - \mathbf{x}_j)/h), \quad i = 1, 2, ..., S \tag{3.17}$$

The kernel regression estimator may now be written in the simplified form as

$$F(\mathbf{x}) = \Sigma_{(i=1,S)} W_{S,i}(\mathbf{x})\, y_i \tag{3.18}$$

Comparing this expression with the generalized output of the FC network given by Eq. (3.2), and bearing in mind that the FC training algorithm makes $u_i = y_i$ for all i, it is clear that the kernel regression approach is equivalent to an FC network where the *k*NN heuristic is invoked, and where the weighting function $W_{S,i}(\mathbf{x})$ is chosen as the membership function, with $k = S$, the sample size. It can be seen that the weighting function is normalized, with

$$\Sigma_{(i=1,S)} W_{S,i}(\mathbf{x}) = 1 \quad \text{for all } \mathbf{x} \tag{3.19}$$

This property of the weighting function $W_{S,i}(\mathbf{x})$ meets the requirement of the membership function in the FC network.

3.10 FC Networks and Radial Basis Function (RBF) Networks

The discussion in Section 3.9 does not specify the choice of the kernel function K. In general, a variety of kernel functions is possible. Indeed, research results have suggested that the choice of the shape of the kernel function K is relatively unimportant compared to the choice of the smoothing parameter h [Hastie and Tibshirani, 1990, pp. 19]. However, both theoretical and practical considerations limit the choice. A widely used kernel function is the multivariate Gaussian distribution:

$$K(\mathbf{x}) = (2\pi)^{-R/2}\exp(-\|\mathbf{x}\|^2/2) \tag{3.20}$$

where R is the dimension of the input vector $\mathbf{x}$. Centering the kernel on a data point, Eq. (3.20) can be written as

$$K((\mathbf{x} - \mathbf{x}_i)/h) = (2\pi h^2)^{-R/2}\exp(-\|\mathbf{x} - \mathbf{x}_i\|^2/2h^2) \quad i = 1, 2, ..., S \tag{3.21}$$

where $\|.\|$ denotes the Euclidean norm of the enclosed vector. Substituting Eq. (3.21) in Eq. (3.16) produces the expression

$$F(\mathbf{x}) = \sum_{(i=1,S)} y_i \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2/2h^2) / \sum_{(j=1,S)} \exp(-\|\mathbf{x} - \mathbf{x}_j\|^2/2h^2) \tag{3.22}$$

This equation resembles the input-output mapping performed by an RBF network [Powell, 1985, 1988, 1992; Moody and Darken, 1989; Poggio and Girosi, 1990; Xu, Krzyzak, and Yuille, 1994]. The parameter h now plays the role of the *spread constant* of the radial basis function. It controls how quickly the value of the function falls to zero as $\mathbf{x}$ moves away from $\mathbf{x}_i$, i.e., the size of its *receptive field* [Wasserman, 1993].

As in Section 3.9, the relationship between the FC network and the RBF network can be seen by defining a weighting function

$$W'_{S,i}(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2/2h^2) / \sum_{(j=1,S)} \exp(-\|\mathbf{x} - \mathbf{x}_j\|^2/2h^2), \tag{3.23}$$

$$i = 1, 2, ..., S$$

and rewriting Eq. (3.22) as

$$F(\mathbf{x}) = \sum_{(i=1,S)} W'_{S,i}(\mathbf{x})\, y_i \tag{3.24}$$

This equation shows that the RBF network can be viewed as an FC network where the kNN heuristic is invoked, and where the weighting function $W'_{S,i}(\mathbf{x})$ is chosen as the membership function, with $k = S$, the sample size.

Despite the above similarity, there are major differences between the FC network and the RBF network:

1. The FC network possesses the instantaneous learning capability while the RBF network does not. The FC network relies on the 1NN heuristic to achieve the input-output mapping specified in the training set. By contrast, since the Gaussian function has non-zero values outside its receptive field, the RBF network as described by Eq. (3.22) must be trained by an iterative process before the desired input-output mapping can be accomplished. There are different paradigms for RBF networks that determined how the training is done [Wasserman, 1993]. One of the simpler methods is to replace the quantities y_i with a set of output weights u_i ($i = 1, 2, ..., S$) and use supervised learning to adjust the values of u_i.
2. When a new vector is presented to an RBF network, the number of training samples that contribute significantly towards the network output varies depending on the distances from the new vector to the respective training samples. By contrast, when the kNN heuristic is invoked in the FC network, the number of training samples that contribute towards the network output is fixed, and equals k.

The RBF network shall be used extensively in this dissertation for performance comparison against the FC network.

3.11 FC Networks and the Bayes Classifier

The Bayes classifier is frequently used as a yardstick against which the performance of other pattern classifiers are measured. From discussion thus far, it is clear that the operation of the FC network as a pattern classifier relies heavily on the nearest-

neighbor concept, fuzzy or otherwise. An FC network switches between a 1NN classifier and a *k*NN classifier depending on the location of the test pattern with respect to the training samples. When the location of the test pattern is such that Rule 1 within the rule base is invoked, the FC network is effectively a 1NN classifier. On the other hand, when Rule 2 is invoked, it is a *k*NN classifier. In this section, the performance of the FC network as 1NN and *k*NN classifiers are analyzed in a statistical framework. In particular, its error rate for each case is expressed in terms of the error rate of the Bayes classifier. Toward this end, it is first necessary to briefly introduce the Bayes decision theory.

3.11.1 Bayes Decision Theory

Bayes decision theory is a fundamental statistical approach to the problem of pattern classification. This approach casts the decision problem in probabilistic terms. It is based on the assumption that all of the relevant probability values are known [Duda and Hart, 1973].

Consider first a *binary* decision problem where an input pattern has to be classified into one of two classes θ_1 and θ_2. Let ω denotes the *state of nature*, with $\omega = \omega_1$ if the input pattern belongs to θ_1, and $\omega = \omega_2$ if it belongs to θ_2. Let the input pattern be characterized by a continuous random variable x whose distribution depends on the state of nature. Let $P(\omega_j)$ be the *a priori probability* that the state of nature is ω_j. Further, let $p(x \mid \omega_j)$ be the *state-conditional probability density function* of x, i.e., the probability density function of x given that the state of nature is ω_j. Then, *Bayes' theorem* states that the *a posteriori* probability $P(\omega_j \mid x)$ is given by

$$P(\omega_j \mid x) = p(x \mid \omega_j)P(\omega_j) / p(x) \tag{3.25}$$

where

$$p(x) = \Sigma_{(j=1,2)}\, p(x \mid \omega_j)P(\omega_j) \tag{3.26}$$

Suppose that both the a priori probabilities $P(\omega_j)$ and the state-conditional density functions $p(x \mid \omega_j)$ are known. Bayes' theorem shows how observing the value of x changes the a priori probabilities $P(\omega_j)$ to the a posteriori probability $P(\omega_j \mid x)$. If $P(\omega_1 \mid x)$ is greater than $P(\omega_2 \mid x)$, then it is likely that the true state of nature is ω_1 and the input pattern should be classified into class θ_1. On the other hand, if $P(\omega_2 \mid x)$ is greater than $P(\omega_1 \mid x)$, then it is likely that the true state of nature is ω_2 and the input pattern should be classified into class θ_2.

The rationale behind this simple decision rule can be justified by computing the probability of error whenever a decision is made. For any value of x,

$$P(\text{error} \mid x) = \begin{cases} P(\omega_1 \mid x) \text{ if } \omega_2 \text{ is chosen} \\ P(\omega_2 \mid x) \text{ if } \omega_1 \text{ is chosen} \end{cases}$$

This decision rule therefore ensures than the probability of error is minimized for each value of x observed. The average probability of error is given by

$$P(\text{error}) = \int_{-\infty}^{\infty} P(\text{error}, x)\, dx$$

$$= \int_{-\infty}^{\infty} P(\text{error} \mid x)p(x)\, dx$$

If $P(\text{error} \mid x)$ is minimized for every x. then the integral is also minimized. Thus, the *Bayes decision rule* for minimizing the probability of error can be stated as follows:

Decide ω_1 if $P(\omega_1 \mid x) > P(\omega_2 \mid x)$; otherwise decide ω_2.

Note that this form of the decision rule emphasizes the role of the a posteriori probabilities. It can be expressed in terms of the a priori and conditional probabilities by noting that, in Eq. (3.25), the denominator $p(x)$ is just a scaling factor to ensure that $P(\omega_1 \mid x) + P(\omega_2 \mid x) = 1$. Therefore, the Bayes decision rule for minimizing the probability of error can be stated in the following equivalent form:

Decide ω_1 if $p(x \mid \omega_1)P(\omega_1) > p(x \mid \omega_2)P(\omega_2)$; otherwise decide ω_2.

The random variable in the above discussion can be generalized from a scalar quantity x to a multivariate *feature vector* $\mathbf{x}$. The restriction of binary decision can also be relaxed to allow a finite set of states of nature. Similarly, the probability of error can be replaced by a more general loss function which specifies how costly each decision is. Let $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$ be the finite set of c states of nature and $A = \{a_1, a_2, ..., a_a\}$ be the finite set of a possible actions. Let $\lambda(a_i \mid \omega_j)$ be the loss incurred for taking action a_i when the true state of nature is ω_j. Then, Bayes' theorem for the generalized case can be stated as

$$P(\omega_j \mid \mathbf{x}) = p(\mathbf{x} \mid \omega_j)P(\omega_j) / p(\mathbf{x}) \tag{3.27}$$

where

$$p(\mathbf{x}) = \Sigma_{(j=1,c)}\, p(\mathbf{x} \mid \omega_j)P(\omega_j) \tag{3.28}$$

Suppose that a particular $\mathbf{x}$ is observed and action a_i taken. If the true state of nature turns out to be ω_j, then the loss incurred is $\lambda(a_i \mid \omega_j)$. Since $P(\omega_j \mid \mathbf{x})$ is the probability that the true state of nature is ω_j when a particular $\mathbf{x}$ is observed, the expected loss associated with taking action a_i is

$$R(a_i \mid \mathbf{x}) = \Sigma_{(j=1,c)} \lambda(a_i \mid \omega_j) P(\omega_j \mid \mathbf{x}) \tag{3.29}$$

where $R(a_i \mid \mathbf{x})$ is known as the *conditional risk* in decision-theoretic terminology. For any value of **x** observed, the expected loss can be minimized by selecting the action that minimizes the conditional risk. Thus, the optimal Bayes decision rule can be stated as follows:

> To minimize the overall risk, compute the conditional risk
>
> $$R(a_i \mid \mathbf{x}) = \Sigma_{(j=1,c)} \lambda(a_i \mid \omega_j) P(\omega_j \mid \mathbf{x})$$
>
> for $i = 1, 2, ..., a$ and select the action a_i for which $R(a_i \mid \mathbf{x})$ is a minimum.

The resulting minimum overall risk is called the *Bayes risk* and it is the best performance that can be achieved by any pattern classifier.

From Bayes decision theory, it is clear that an optimal pattern classifier can be designed if the a priori probabilities $P(\omega_j)$ and the state-conditional densities $p(\mathbf{x} \mid \omega_j)$ are known. However, in real-world pattern classification problems, these probabilities and densities are seldom known. Nevertheless, the Bayes classifier is important as a common reference for comparing the performance of other pattern classifiers. It is in this spirit that the performance of the FC network as a 1NN and a *k*NN classifier is analyzed in the next two subsections.

3.11.2 FC Network as a Single-Nearest-Neighbor Classifier

The nearest-neighbor classification concept is based on the reasonable assumption that input patterns that are close together will have the same output classification, or at least will have almost the same a posteriori probability distributions on their respective

classifications. Consider a set of n independently drawn labeled samples $(\mathbf{x}_1, \theta_1)$, $(\mathbf{x}_2, \theta_2)$, ..., $(\mathbf{x}_n, \theta_n)$ where θ_i may correspond to any of the c states of nature $\omega_1, \omega_2, ..., \omega_c$. Let $d(\mathbf{x}_i, \mathbf{x}_j)$ denote the vector distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ in some appropriate metric, such as the Euclidean metric. Then $\mathbf{x}'_n \in \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ is said to be the nearest neighbor of $\mathbf{x}$ if

$$\min d(\mathbf{x}_i, \mathbf{x}) = d(\mathbf{x}'_n, \mathbf{x}) \qquad i = 1, 2, ..., n$$

The 1NN classifier assigns $\mathbf{x}$ to the class θ'_n that corresponds to $\mathbf{x}'_n$. To simplify the following analysis, it is assumed that the number of samples is large. The error rate of the 1NN classifier can be obtained by first computing the *conditional* average probability of error $P(e \mid \mathbf{x})$, where the averaging is with respect to the samples. The unconditional error rate $P(e)$ can then be found by averaging $P(e \mid \mathbf{x})$ over all $\mathbf{x}$ [Duda and Hart, 1973]:

$$P(e) = \int P(e \mid \mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{3.30}$$

Recall that the Bayes decision rule minimizes $P(e)$ by minimizing $P(e \mid \mathbf{x})$ for every $\mathbf{x}$. Let $P^*(e \mid \mathbf{x})$ be the minimum possible value of $P(e \mid \mathbf{x})$. Then the minimum possible value of $P(e)$, i.e., the Bayes error rate, denoted by P^*, is

$$P^* = \int P^*(e \mid \mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{3.31}$$

The objective of this subsection is to express the bounds of the 1NN classifier error rate $P(e)$ in terms of the Bayes error rate P^*. Since the Bayes classifier is the optimal classifier, $P(e)$ is bounded below by P^* itself.

Suppose that a particular $\mathbf{x}$ is observed and it is assigned a label θ by the 1NN classifier. If the sample nearest to $\mathbf{x}$ is $\mathbf{x}'_n$, then $\theta = \theta'_n$. However, if the true label for $\mathbf{x}$

is not θ'_n, then the classification is in error. The conditional probability of error is therefore given by

$$\begin{aligned} P_n(e \mid \mathbf{x}, \mathbf{x}'_n) &= P(\theta \neq \theta'_n \mid \mathbf{x}, \mathbf{x}'_n) \\ &= 1 - \Sigma_{(i=1,c)} P(\theta = \omega_i, \theta'_n = \omega_i \mid \mathbf{x}, \mathbf{x}'_n) \\ &= 1 - \Sigma_{(i=1,c)} P(\omega_i \mid \mathbf{x}) P(\omega_i \mid \mathbf{x}'_n) \end{aligned} \tag{3.32}$$

If the number of samples is large, then $\mathbf{x}$ is very close to $\mathbf{x}'_n$, such that

$$P(\omega_i \mid \mathbf{x}) \simeq P(\omega_i \mid \mathbf{x}'_n) \tag{3.33}$$

Eq. (3.32) then reduces to the form

$$\lim_{n\to\infty} P_n(e \mid \mathbf{x}) = 1 - \Sigma_{(i=1,c)} P^2(\omega_i \mid \mathbf{x}) \tag{3.34}$$

Averaging this over all $\mathbf{x}$ gives the large-sample error rate of the 1NN classifier

$$\begin{aligned} P &= \lim_{n\to\infty} P_n(e) \\ &= \lim_{n\to\infty} \int P_n(e \mid \mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \\ &= \int \{1 - \Sigma_{(i=1,c)} P^2(\omega_i \mid \mathbf{x})\} p(\mathbf{x})\, d\mathbf{x} \end{aligned} \tag{3.35}$$

The upper bound for P corresponds to the minimum of $\Sigma_{(i=1,c)} P^2(\omega_i \mid \mathbf{x})$. This summation term can be split into two parts:

$$\Sigma_{(i=1,c)} P^2(\omega_i \mid \mathbf{x}) = P^2(\omega_m \mid \mathbf{x}) + \Sigma_{(i\neq m)} P^2(\omega_i \mid \mathbf{x}) \tag{3.36}$$

The first term on the right can be written in terms of $P^*(e \mid \mathbf{x})$ by defining ω_m such that, when $\mathbf{x}$ is observed, the maximum of all $P(\omega_i \mid \mathbf{x})$ occurs for some $i = m$ so that the Bayes classifier will select ω_m. The error rate of the Bayes classifier is therefore

$$P^*(e \mid \mathbf{x}) = 1 - P(\omega_m \mid \mathbf{x}) \tag{3.37}$$

The second term on the right side of Eq. (3.36) can also be expressed in terms of $P^*(e \mid \mathbf{x})$ by noting that

$$\Sigma_{(i\neq m)}P(\omega_i \mid \mathbf{x}) = 1 - P(\omega_m \mid \mathbf{x}) = P^*(e \mid \mathbf{x}) \tag{3.38}$$

Further, due to the effect of squaring, $\Sigma_{(i\neq m)}P^2(\omega_i \mid \mathbf{x})$ is minimized if all $P(\omega_i \mid \mathbf{x})$ in the summation have the same value, with each equal to $P^*(e \mid \mathbf{x})/(c\text{-}1)$. Therefore,

$$\min(\Sigma_{(i\neq m)}P^2(\omega_i \mid \mathbf{x})) = (c\text{-}1)\{P^*(e \mid \mathbf{x})/(c\text{-}1)\}^2 \tag{3.39}$$

i.e.,

$$\Sigma_{(i\neq m)}P^2(\omega_i \mid \mathbf{x}) \geq P^{*2}(e \mid \mathbf{x})/(c\text{-}1) \tag{3.40}$$

Substituting Eq. (3.37) and Eq. (3.40) in Eq. (3.36) gives

$$\Sigma_{(i=1,c)}P^2(\omega_i \mid \mathbf{x}) \geq (1 - P^*(e \mid \mathbf{x}))^2 + P^{*2}(e \mid \mathbf{x})/(c\text{-}1) \tag{3.41}$$

Simplifying and rearranging terms gives

$$1 - \Sigma_{(i=1,c)}P^2(\omega_i \mid \mathbf{x}) \leq 2P^*(e \mid \mathbf{x}) - cP^{*2}(e \mid \mathbf{x})/(c\text{-}1) \tag{3.42}$$

Substituting Eq. (3.42) in Eq. (3.35) results in

$$P \leq \int \{\, 2P^*(e \mid \mathbf{x}) - cP^{*2}(e \mid \mathbf{x})/(c\text{-}1)\}p(\mathbf{x})\, d\mathbf{x} \tag{3.43}$$

which leads to the conclusion $P \leq 2P^*$ simply by using Eq. (3.31) and ignoring the second term on the right. In other words, if the number of samples is large, the error rate of the 1NN classifier is never worse than twice that of the Bayes classifier.

A tighter upper bound for the 1NN classifier error rate can be obtained by noting that

$$\begin{aligned}\mathrm{Var}[P^*(e \mid \mathbf{x})] &= \int [P^*(e \mid \mathbf{x}) - P^*]^2\, p(\mathbf{x})\, d\mathbf{x} \\ &= \int P^{*2}(e \mid \mathbf{x})p(\mathbf{x})\, d\mathbf{x} + P^{*2}\int p(\mathbf{x})\, d\mathbf{x} - 2P^*\int P^*(e \mid \mathbf{x})p(\mathbf{x})\, d\mathbf{x}\end{aligned}$$

Using Eq. (3.31) and the fact that $\int p(\mathbf{x})\, d\mathbf{x} = 1$, this expression can be simplified into

$$\mathrm{Var}[P^*(e \mid \mathbf{x})] = \int P^{*2}(e \mid \mathbf{x})p(\mathbf{x})\, d\mathbf{x} - P^{*2}$$

Since $\mathrm{Var}[P^*(e \mid \mathbf{x})] \geq 0$, it can be concluded that

$$\int P^{*2}(e \mid \mathbf{x})p(\mathbf{x})\, d\mathbf{x} \geq P^{*2}$$

Substituting this result in Eq. (3.43) and appending the lower bound gives

$$P^* \leq P \leq P^*(2 - cP^*/(c\text{-}1)) \tag{3.44}$$

The variation in the upper and lower bounds of P as a function of P^* is shown graphically in Figure 13. The curve represents the upper bound while the straight line represents the lower bound, i.e., the Bayes rate P^*, which can be anywhere between 0 and $(c - 1)/c$. The two bounds meet at these two extreme points. When P^* is small, the upper bound is approximately twice P^*. In general, the 1NN classifier error rate falls in the area bounded by the curve and the straight line.

3.11.3 FC Network as a *k*-Nearest-Neighbor Classifier

The discussion on the performance of the 1NN classifier can be extended to the kNN classifier. The analysis below also serves to provide some insight for choosing the value of k.

Consider a two-class problem where the classification is done by voting. To avoid the possibility of a tie in the voting process, k is assumed to be odd. Suppose that when a particular $\mathbf{x}$ is observed, more than half of its k nearest neighbors have the label θ_i so that the kNN classifier assigns $\mathbf{x}$ the label θ_i $(i = 1, 2)$. If the true state of nature turns out to be other than ω_i, then the classification is in error. The conditional probability of error $P(e \mid \mathbf{x})_k$ is thus given by

$$P(e \mid \mathbf{x})_k = P(\omega_1 \mid \mathbf{x})\Sigma_{j=0,(k-1)/2}\binom{k}{j}P(\omega_1 \mid \mathbf{x})^j P(\omega_2 \mid \mathbf{x})^{k-j}$$

$$+ P(\omega_2 \mid \mathbf{x})\Sigma_{j=0,(k-1)/2}\binom{k}{j}P(\omega_2 \mid \mathbf{x})^j P(\omega_1 \mid \mathbf{x})^{k-j} \tag{3.45}$$

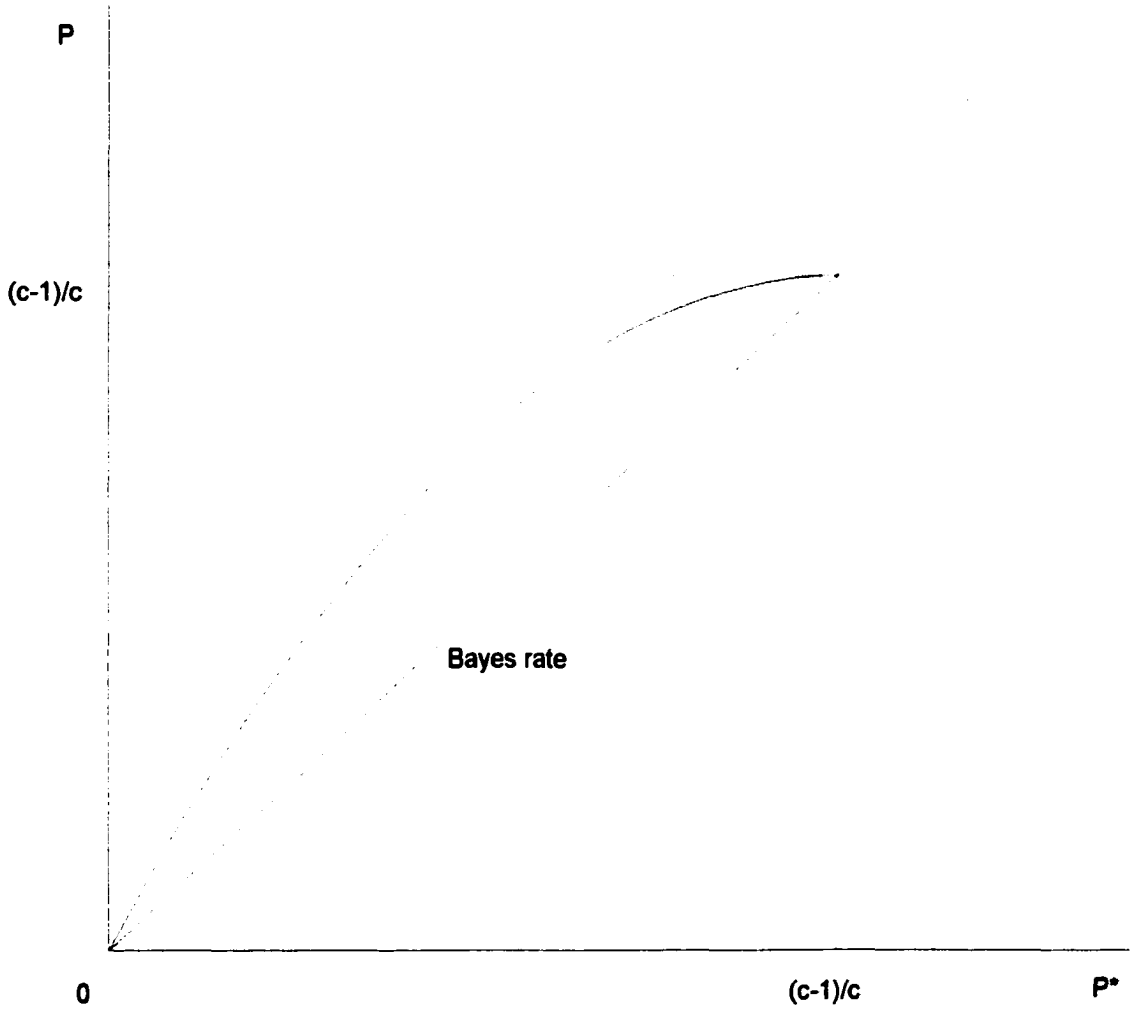

Fig. 13: Bounds on 1NN classifier error rate

where the first and second terms are the conditional error when the true state of nature is ω_1 and ω_2 respectively. The notation $\binom{k}{j}$ refers to the binomial combination of choosing j items from k possibilities. Since $P(\omega_1 | \mathbf{x}) + P(\omega_2 | \mathbf{x}) = 1$, and by using the fact that choosing "less than half" from one class is equivalent to choosing "more than half" from the other for this two-class problem, Eq. (3.45) can be rewritten in terms of $P(\omega_1 | \mathbf{x})$ alone as

$$P(e | \mathbf{x})_k = P(\omega_1 | \mathbf{x}) \sum_{j=0,(k-1)/2} \binom{k}{j} P(\omega_1 | \mathbf{x})^j [1 - P(\omega_1 | \mathbf{x})]^{k-j}$$

$$+ [1 - P(\omega_1 | \mathbf{x})] \sum_{j=(k+1)/2,k} \binom{k}{j} P(\omega_1 | \mathbf{x})^j [1 - P(\omega_1 | \mathbf{x})]^{k-j} \quad (3.46)$$

or in terms of $P(\omega_2 | \mathbf{x})$ alone as

$$P(e | \mathbf{x})_k = P(\omega_2 | \mathbf{x}) \sum_{j=0,(k-1)/2} \binom{k}{j} P(\omega_2 | \mathbf{x})^j [1 - P(\omega_2 | \mathbf{x})]^{k-j}$$

$$+ [1 - P(\omega_2 | \mathbf{x})] \sum_{j=(k+1)/2,k} \binom{k}{j} P(\omega_2 | \mathbf{x})^j [1 - P(\omega_2 | \mathbf{x})]^{k-j} \quad (3.47)$$

Further, since the Bayes error rate $P^* = \min[P(\omega_1 | \mathbf{x}), P(\omega_2 | \mathbf{x})]$, $P(e | \mathbf{x})_k$ can be expressed in terms of P^* as

$$P(e | \mathbf{x})_k = P^* \sum_{j=0,(k-1)/2} \binom{k}{j} (P^*)^j (1 - P^*)^{k-j}$$

$$+ (1 - P^*) \sum_{j=(k+1)/2,k} \binom{k}{j} (P^*)^j (1 - P^*)^{k-j} \quad (3.48)$$

which follows from Eq. (3.46) if $P(\omega_1 | \mathbf{x}) < P(\omega_2 | \mathbf{x})$, and from Eq. (3.47) if $P(\omega_2 | \mathbf{x}) < P(\omega_1 | \mathbf{x})$. Simplifying Eq. (3.48) gives

$$P(e | \mathbf{x})_k = \sum_{j=0,(k-1)/2} \binom{k}{j} (P^*)^{j+1} (1 - P^*)^{k-j}$$

$$+ (1 - P^*) \sum_{j=0,(k-1)/2} \binom{k}{j} (1 - P^*)^j (P^*)^{k-j}$$

$$= \sum_{j=0,(k-1)/2} \binom{k}{j} [(P^*)^{j+1}(1 - P^*)^{k-j} + (P^*)^{k-j}(1 - P^*)^{j+1}] \quad (3.49)$$

The variation of $P(e \mid \mathbf{x})_k$ as a function of P^* for various values of k is shown in Figure 14. It is seen that as k increases, the large-sample performance of the kNN classifier approaches that of the Bayes classifier. However, the value of k must be kept small so that all the k nearest neighbors are very close to $\mathbf{x}$ to justify the assumption that their a posteriori probabilities are very similar to that of $\mathbf{x}$. Thus, in practice, the value of k is typically chosen to be a small fraction of the sample size.

3.12 Summary

In this chapter, the theory behind the FC network is developed. It is shown that the network can be trained with just two passes of the training samples. The first pass assigns the synaptic weights for the input and output layers. The second pass determines the radius of generalization r for each training sample. The network exhibits fuzziness in two regards: (1) by fuzzification of the location of each training vector in the input space; and (2) by assigning fuzzy memberships of output classes to new input vectors. The notion of radius of generalization for each training vector provides a basis for the network to switch between a 1NN classifier and a kNN classifier during generalization. The network behaves like a 1NN classifier when the input vector falls within the area of generalization of a training vector, and a kNN classifier otherwise. This enables the network to benefit from the strength of both classifiers.

The operation of the FC network is then viewed from various perspectives. It is shown that the transformation performed by an FC network is consistent with Cover's

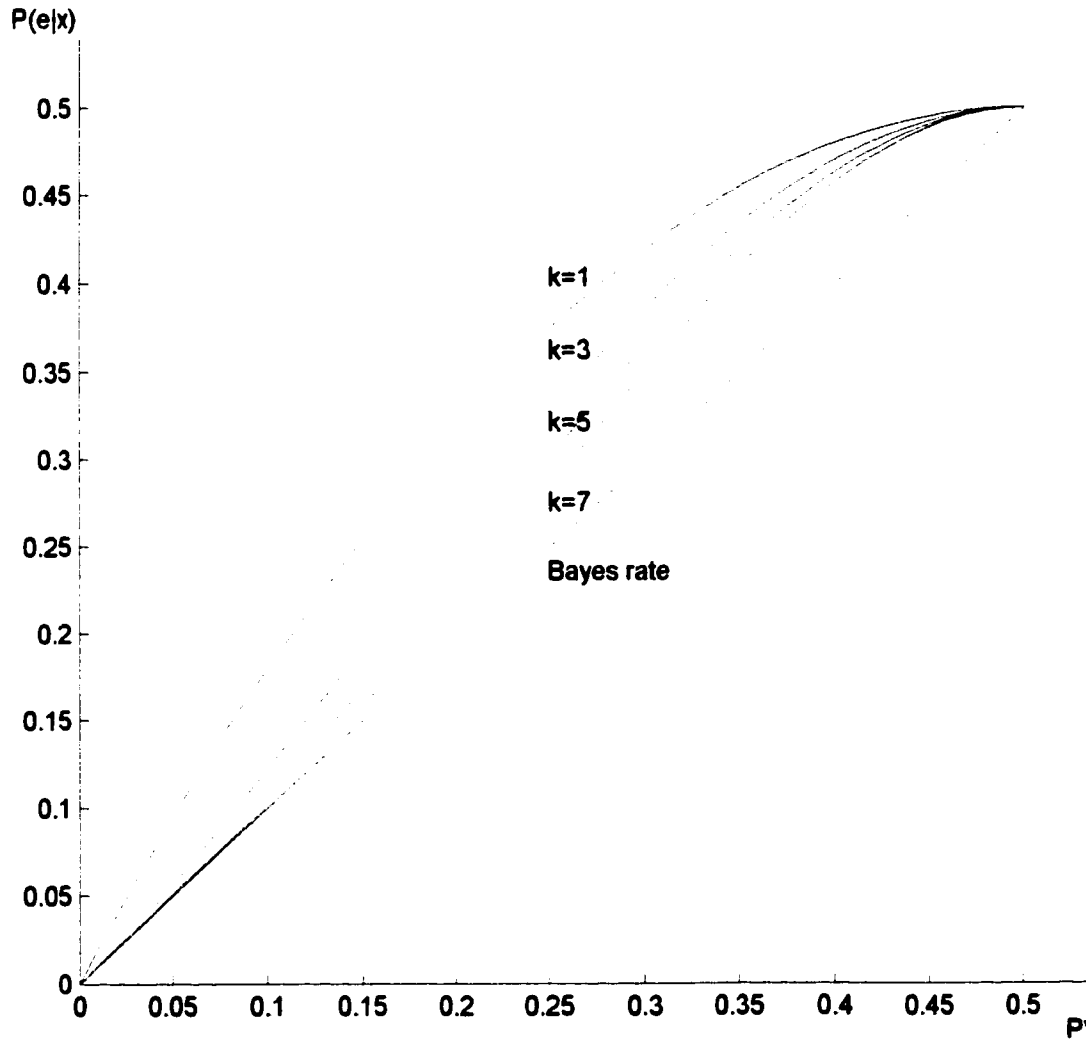

Fig. 14: Variation of kNN classifier error rate as a function of k and P^*

theorem on separability of patterns. This is illustrated with an example featuring the classic Exclusive-OR problem. It is also argued that the FC network meets the specification of traditional curve fitting in its strict sense in that the function implemented by the FC network is constrained to pass through all data points given in the training set. It is further shown that the FC network operating as a *k*NN classifier can be viewed as kernel regression if $k = S$ and the weighting function is chosen as the membership function. Along a similar line of argument, it is shown that the FC network operating as a *k*NN classifier can be made to behave like an RBF network if $k = S$ and the Gaussian distribution function is chosen as the membership function.

The performance of the FC network as a 1NN and a *k*NN classifier is then cast in probabilistic terms and in relation to the Bayes classifier. Unfortunately, while the analysis is relatively straightforward for a two-class problem where classification is done by voting, the more general cases where classification is done by fuzzy membership and where the problem involves multiple classes are not so amenable to a similar kind of treatment. The discussion on the FC network as a *k*NN classifier also provides some insight regarding the choice of the value of *k*. While theoretical analysis suggests that *k* should be made large to improve performance, practical consideration resulting from finite sample size dictates that *k* be kept to a small fraction of the sample size. It shall be seen in the next two chapters that the choice of a small *k* is confirmed in experiments involving time series prediction and pattern classification.

Chapter 4

Function Approximation and Time Series Prediction Using Fuzzy Classification Neural Networks

A major application area for neural networks trained by supervised learning is function approximation where a neural network attempts to approximate an unknown function by learning the mapping between the function input and output from a set of training samples. The network is then able to generate the appropriate response corresponding to any new input that is not in the original training set. Time series prediction falls into this domain. In time series prediction, a neural network learns the characteristics of the time series from historical data and then attempts to predict its future values based on some recently collected data. This chapter discusses the FC network in the context of function approximation and time series prediction.

4.1 Function Approximation

Consider a system with nonlinear input-output mapping described by the functional relationship

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

where vector $\mathbf{x}$ is the input, vector $\mathbf{y}$ is the output, and the vector-valued function $\mathbf{f}(\cdot)$ is assumed to be unknown. What is known, however, is a set of observed values $\mathbf{y}_i$ ($i = 1, 2, ..., S$) at the output of the system corresponding to a set of input states $\mathbf{x}_i$ of the

system. The task of the neural network is to implement a function $\mathbf{F}(\mathbf{x})$ which approximates the unknown function $\mathbf{f}(\mathbf{x})$ such that the input-output mapping realized by $\mathbf{F}(\mathbf{x})$ is *close* to $\mathbf{f}(\mathbf{x})$ in a Euclidean sense over all inputs, i.e.,

$$\|\mathbf{F}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\| < \varepsilon \qquad \text{for all } \mathbf{x}$$

where ε is a small positive number. The input-output pairs $(\mathbf{x}_i, \mathbf{y}_i)$ are the training samples from which the network can learn the input-output relation. The approximation error ε can be made small enough for the task in hand by using a sufficiently large sample size S.

Two important applications where the ability of a neural network to perform function approximation is put to good use are *system identification* and *inverse system modeling.*

4.1.1 System Identification

Figure 15 shows the block diagram of a typical arrangement for system identification, where the unknown system is to be approximated by the neural network. It is assumed that the unknown system is time invariant. The network is first trained using a set of training data collected from the system. It is then connected in parallel with the unknown system and the input $\mathbf{x}_i$ varied over the entire operating range of the system. At each operating point under test, the system output $\mathbf{y}_i$ and neural network output $\mathbf{z}_i$ are passed through a comparator to produce the error signal $\mathbf{e}_i$. If the error signal is found to be beyond an acceptable level at some operating point j, the input-output pair $(\mathbf{x}_j, \mathbf{y}_j)$ corresponding to that operating point is used to provide further training to the network. For an instantaneous learning neural network such as the FC

network, this is accomplished by adding a hidden neuron to learn ($\mathbf{x}_i$, $\mathbf{y}_i$). This process is repeated until the error signal is small enough for all operating points of the system. The neural network is then taken to be an acceptable model of the unknown system.

4.1.2 Inverse System Modeling

This is the inverse of the previous problem. Given a time-invariant system whose input-output mapping is described by $\mathbf{y} = \mathbf{f}(\mathbf{x})$, the task of the neural network is to implement an inverse system that produces the vector $\mathbf{x}$ in response to the vector $\mathbf{y}$. The inverse system is thus described by the relationship

$$\mathbf{x} = \mathbf{f}^{-1}(\mathbf{y})$$

where $\mathbf{f}^{-1}(\cdot)$ is understood to represent an inverse function (i.e., reverse mapping) rather than the reciprocal of $\mathbf{f}(\cdot)$. In most practical situations, the function $\mathbf{f}(\cdot)$ is too complicated to allow a straightforward formulation of the inverse function $\mathbf{f}^{-1}(\cdot)$.

Figure 16 shows the block diagram of a typical arrangement for inverse system modeling. The neural network is first trained using a set of training data collected from the system. However, the roles of input and output data are reversed: system output $\mathbf{y}_i$ becomes the training input to the neural network and system input $\mathbf{x}_i$ becomes the target output for the neural network. The network is then connected in *series* with the system such that output from the system is fed as input to the network. Neural network output $\mathbf{z}_i$ is compared against $\mathbf{x}_i$ to generate the error signal $\mathbf{e}_i$ over the entire operating range of the system. As before, the network is trained until the error signal is small enough over the full operating range of the system.

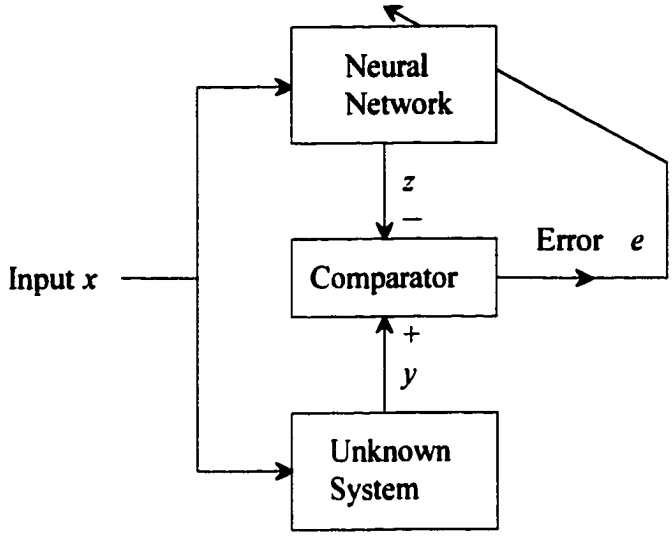

Fig. 15: System identification

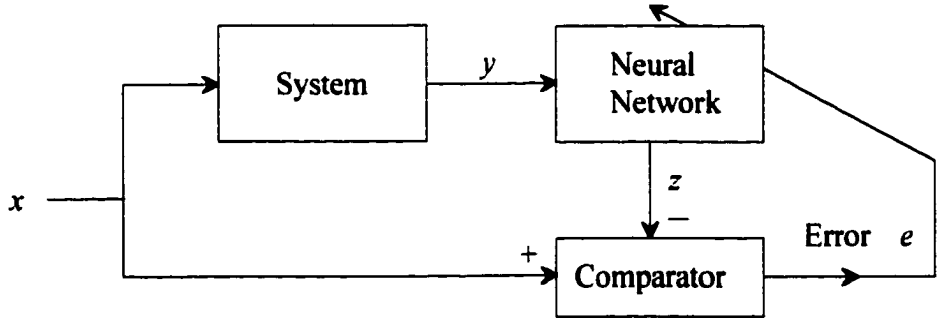

Fig. 16: Inverse system modeling

System identification and inverse system modeling problems described above are commonly encountered in industrial control and engineering fields. Neural network has increasingly been used as an alternative to conventional techniques by control engineers for solving such problems.

4.2 Time Series Prediction

In time series prediction, a set of historical data, usually indexed by time, is used to train a neural network. The network can then predict future values of the time series based on several values of the time series observed most recently. Time series prediction using neural networks finds widespread application in engineering, financial and economic fields. Some examples are electrical load demand forecast, traffic volume forecast, and prediction of stock prices, currency rates, and interest rates. In this chapter, the performance of the FC network shall be tested using two benchmark time series with different characteristics. The first is the Henon Map time series, which is very volatile, while the second is the Mackey-Glass time series, which changes direction more slowly.

4.2.1 Henon Map Time Series Prediction Using FC Network

The Henon map [Henon, 1976; Thompson and Stewart, 1986; Stokbro and Umberger, 1992] is defined by a two-dimensional system of equations

$$x(t) = 1 - ax^2(t\text{-}1) + y(t\text{-}1)$$

$$y(t) = bx(t\text{-}1)$$

which can be combined to generate a one-dimensional time series

$$x(t) = 1 - ax^2(t\text{-}1) + bx(t\text{-}2)$$

A chaotic regime occurs for $a = 1.4$ and $b = 0.3$ [Azoff, 1994]. This equation is used to generate a total of 554 points out of which 500 training samples are derived from the first 504 points while the remaining 50 points are used for out-of-sample testing of the network after training. Due to the volatile nature of this time series, the number of testing points is kept small at 50 to improve the readability of the graphical display and printouts.

In this experiment, the window size is set to 4 and prediction is made for one point ahead. In other words, each training sample is formed by a sliding window consisting of four consecutive points along the time series as input with the fifth point as the target output. For example, the first training sample consists of $x(1)$, $x(2)$, $x(3)$ and $x(4)$ as input while the target output is $x(5)$. The second training sample uses $x(2)$, $x(3)$, $x(4)$ and $x(5)$ as input while the target output is $x(6)$, and so on. The last training sample therefore consists of $x(500)$, $x(501)$, $x(502)$ and $x(503)$ as input with $x(504)$ as the target output. The FC network architecture needed is thus 4-500-1.

The out-of-sample testing points are similarly organized into test vectors. For example, the first test vector is made up of $x(501)$, $x(502)$, $x(503)$ and $x(504)$. The actual output corresponding to this test vector is $x(505)$. After the network has been trained using the sample set, the fifty test vectors are presented one at a time and the network is required to predict the output corresponding to each of the test vectors. The predicted output is then compared against the actual output and the difference between the two is used to compute the total prediction error expressed in terms of a sum-of-squared error (SSE) cumulated over the fifty test points. Table 1 shows the prediction error for various values of k.

Table 1: Henon map time series prediction using FC network

k	SSE
1	0.009054
2	0.007567
3	0.003381
4	0.002971
5	**0.002481**
6	0.002486
7	0.003401

This table shows that the best performance is obtained when $k = 5$. Figure 17 shows a plot of the predicted output against the actual data points spanning the points $x(505)$ to $x(554)$. In the figure, the actual data points are represented by "*" and connected by a solid line while the predicted points are marked by "o" and connected by a dotted line. It can be seen that the predicted points match the actual data points very well. In particular, all turning points in the time series have been correctly predicted. Points before $x(505)$ are not shown since the match between actual data and network output is exact, i.e., training error is zero in a FC network.

The close match between the predicted and the actual data points shown in Figure 17 is visually comparable to that of a similar experiment reported in the literature [Figure 6.1, page 96, Azoff, 1994] which was obtained using a BP network implementing the quasi-Newton inexact line search technique known as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) gradient search method. A more thorough comparison with CC4, RBF and BP networks shall be conducted below.

(A) Comparison with CC4 Network

Since CC4 uses binary input and output, the time series data must first be digitized into binary vectors. To ensure good performance from the CC4 network, it is necessary to

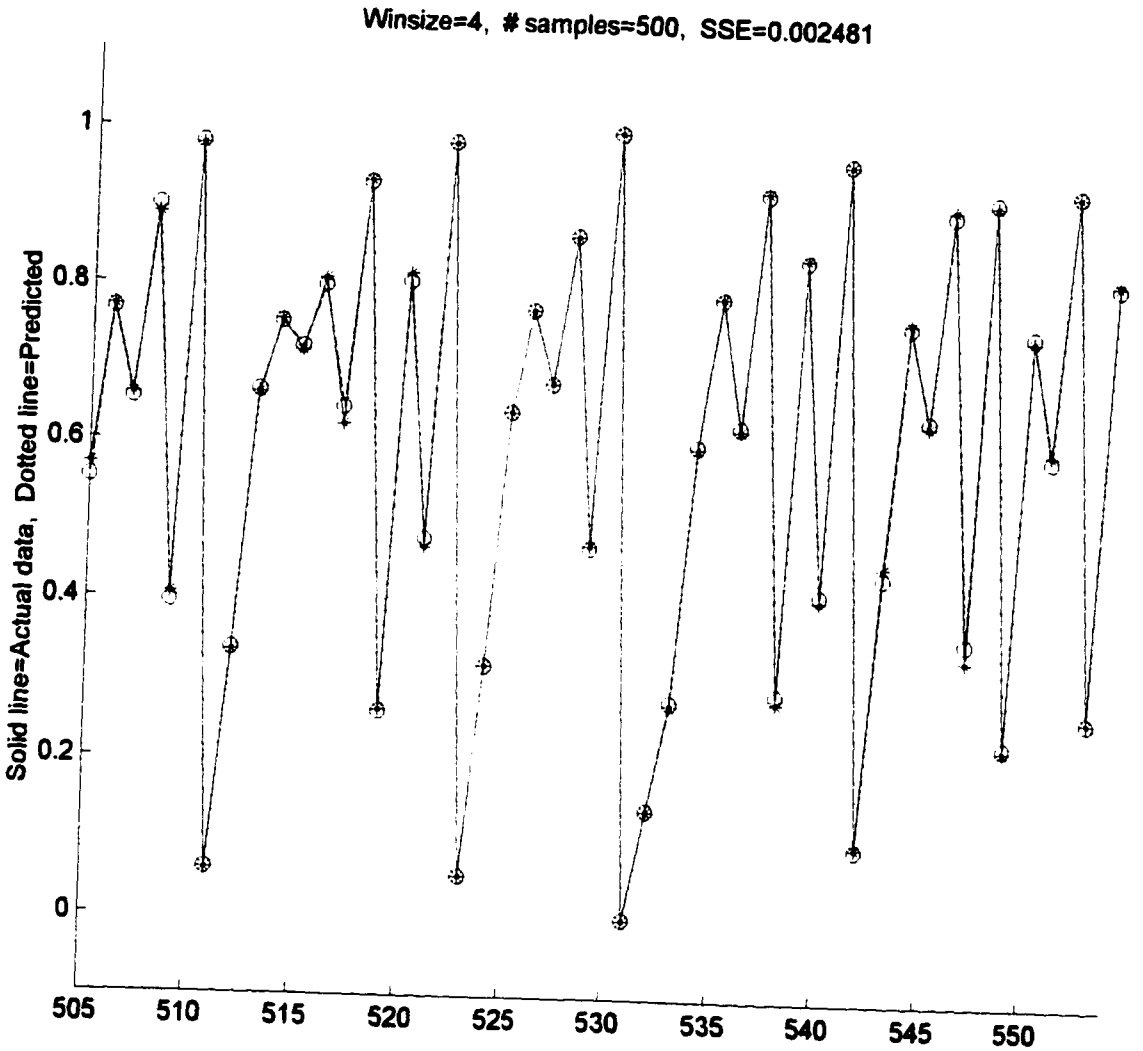

Fig. 17: Henon map time series prediction using FC (4-500-1), $k = 5$

find the optimum quantization level, which is the number of bits used to digitize the data, and the optimum radius of generalization *r* for the network. Quantization levels from 20 to 100 bits in steps of 10 bits are tested. For each quantization level, the network is trained with different values of *r* and the prediction error (SSE) noted. Table 2 shows the result obtained.

Table 2: Henon map time series prediction using CC4 network

Quantization level (bits)	**Best *r***	**SSE**
20	3	1.0309
30	3	0.6168
40	3	0.2974
50	5	0.4247
60	**4**	**0.1462**
70	7	0.2985
80	8	0.3946
90	8	0.2436
100	9	0.2344

The lowest SSE occurs for a quantization level of 60 and a radius of generalization of 4. Since each network input consists of four data points, the total length of the input vector is 240. The CC4 network architecture needed for this time series is thus 241-500-60. Figure 18 shows the plot of the predicted output against the actual data points. Comparing it with that shown in Figure 17, it can be seen that the FC network, with an SSE of 0.00248, gives a better performance than this CC4 network, which has an SSE of 0.146.

(B) Comparison with RBF Network

The Matlab Neural Network Toolbox comes with two functions for designing and training RBF networks: **solverbe** and **solverb**. **solverbe** creates a network with as

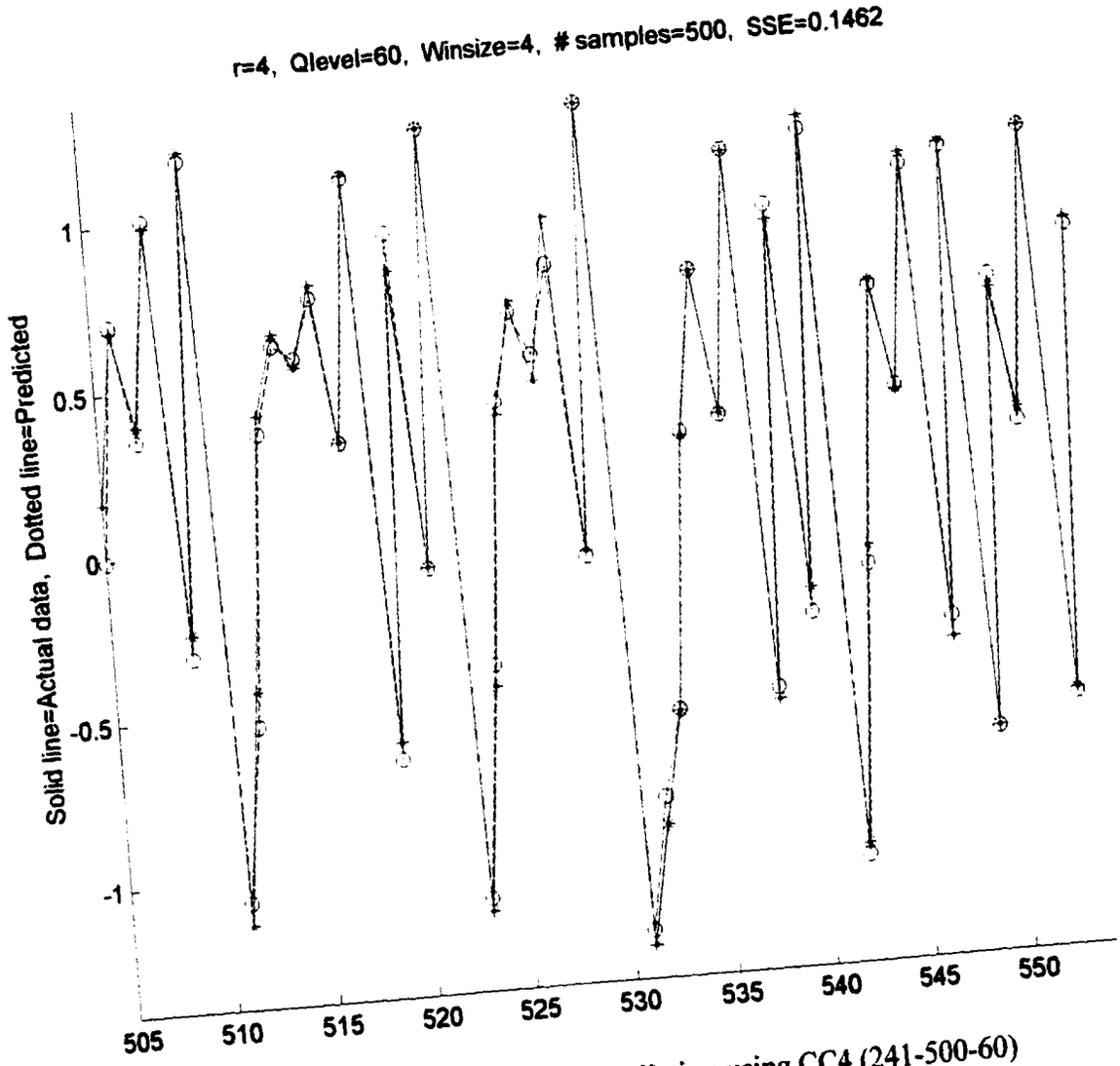

Fig. 18: Henon map time series prediction using CC4 (241-500-60)

many hidden neurons as there are training samples to be learned. The result is an *exact* network with no error for the input/target vectors used to design it, i.e., the training set [Matlab, 1995]. **solverb**, on the other hand, creates a network by adding hidden neurons one at a time. At each iteration, a hidden neuron is added to learn the training input/target vector that would result in the greatest reduction in network error. Training stops when the user-specified error goal is achieved, or when the maximum number of hidden neurons specified by the user is exhausted. This creates a parsimonious network but the iterative process involved is very time consuming.

For a fair comparison, the exact version, **solverbe**, shall be used here. The RBF network architecture needed to learn this time series is thus 4-500-1, which is the same as that for the FC network. To ensure good performance from the RBF network, a suitable *spread constant* must first be found. The spread constant is the distance from where the radial basis function has a peak value of 1.0 to where it has a value of 0.5. It controls the width of the receptive field of each hidden neuron in the input space. The RBF network is trained using various spread constants starting from 0.01 and the prediction error (SSE) summed over all test points noted for each case. Table 3 shows part of the result obtained.

Table 3: Henon map time series prediction using RBF network

Spread Constant	SSE
0.05	1
0.1	0.92
0.15	0.06
0.2	**0.00096**

It can be seen that a spread constant of 0.2 gives excellent performance. A plot of the predicted points versus the actual data points is depicted in Figure 19. It can be seen that the performance of this RBF network is better than that of the FC network. However, Table 3 also reveals that the performance of the RBF network is highly sensitive to the choice of value for the spread constant. For example, if the spread constant changes from 0.2 to 0.1, the SSE increases from 0.00096 to 0.92, a thousand-fold increase.

(C) Comparison with BP Network

The Matlab Neural Network Toolbox comes with four functions for use in designing and training BP networks: an initialization function called **initff** and three training functions called **trainbp, trainbpx** and **trainlm**. All functions require the user to decide in advance the number of hidden layers and the number of hidden neurons in each layer. **initff** takes the training set and initializes a network with suitable starting weights and biases. It is important that the training set contains the expected maximum and minimum values of all input variables so that the best starting weights and biases can be calculated. **trainbp** is the simplest of the three backpropagation functions. It uses the gradient descent algorithm without momentum feature and it requires the user to specify the learning rate, which affects the size of the changes made to the weights and biases at each iteration. Picking the learning rate is a challenge: a small learning rate results in a long training time while a large learning rate leads to unstable learning. **trainbpx** improves on **trainbp** by introducing the momentum feature as well as incorporating the *adaptive* learning rate. The adaptive learning rate starts with an

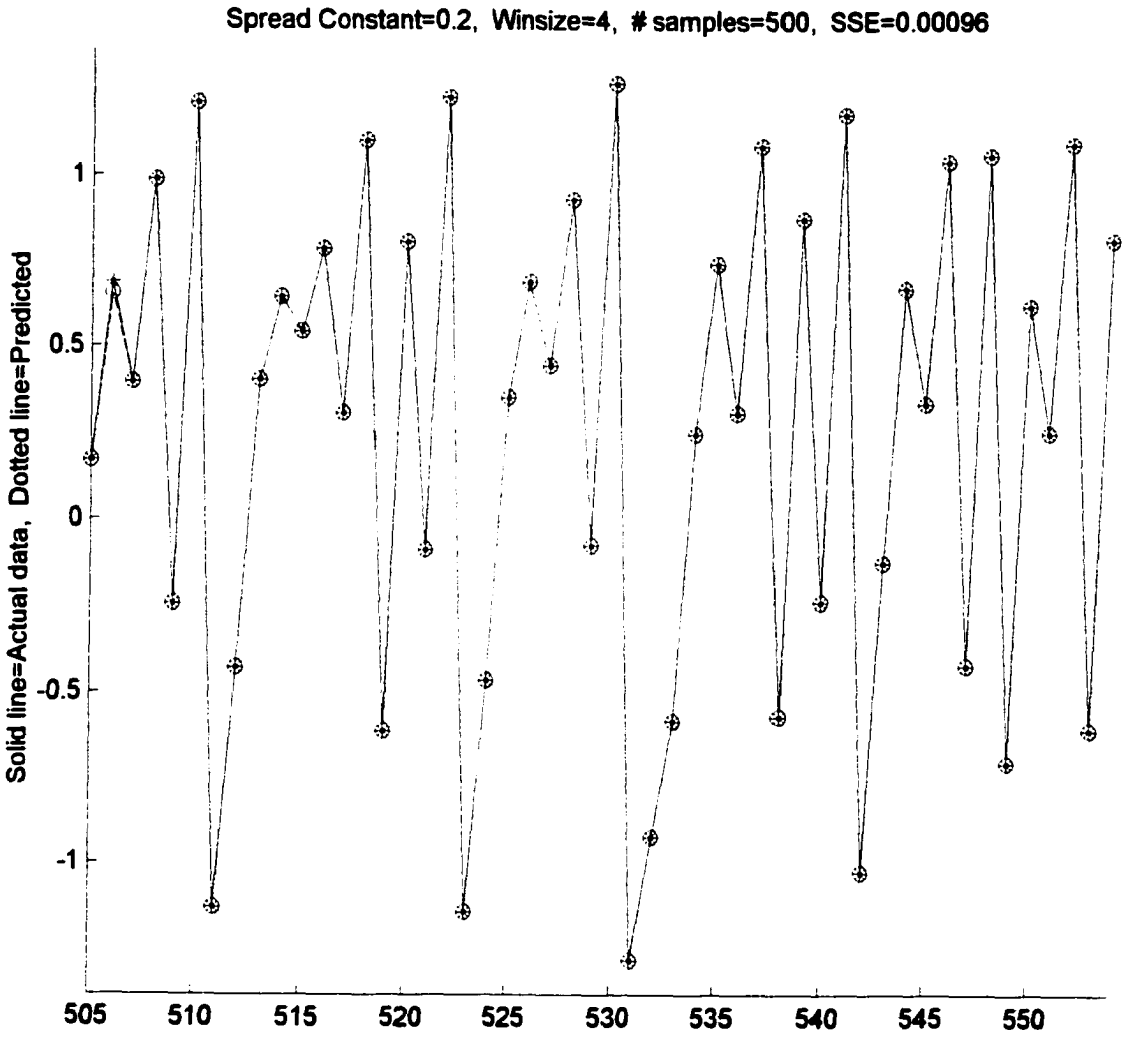

Fig. 19: Henon map time series prediction using RBF (4-500-1)

initial value which is then increased or decreased depending on whether the gradient search is heading in the correct direction. This technique helps keep learning fast but stable.

While both **trainbp** and **trainbpx** make use of the gradient descent technique, **trainlm** uses the Levenberg-Marquardt optimization algorithm which is many times faster. However, it requires a lot of memory. Typically, the amount of memory required by **trainlm** is SxQ times that required by **trainbp** [Matlab, 1995], where S is the number of output neurons and Q is the number of training samples. This enormous memory requirement prohibits the use of **trainlm** except for very small problems. Thus, all experiments on BP in this dissertation shall be done using **trainbpx**. For simplicity, BP networks with one hidden layer shall be used.

As with the CC4 and RBF networks, to ensure good performance from a BP network, it is necessary to find the optimum number of hidden neurons needed to learn the time series. This is done by searching through different network configurations. For each configuration, the network is trained ten times, each with a different set of initial weights and biases. The average prediction error over the ten runs are calculated. The number of training epochs for each run is deliberately kept small at 1000. The reason for this is that a network with insufficient hidden neurons would either get trapped in local minimum easily or learn very slowly. Thus, with a small number of training epochs, only networks with sufficient number of hidden neurons can produce low prediction errors averaged over ten runs. Table 4 shows part of the search result.

Table 4: Henon map time series prediction using BP network

Network Configuration	Average SSE over 10 runs
4-5-1	0.0588
4-6-1	0.0474
4-7-1	0.0328
4-8-1	0.0325
4-9-1	**0.0316**
4-10-1	0.0320
4-11-1	0.0421

It can be seen that a network with nine hidden neurons gives the lowest average prediction error. This network is then given more training. Figure 20 depicts a typical plot of the predictions made by this network after being trained for 3000 epochs. Comparing it with Figure 17, it is clear that the FC network outperforms this BP network.

4.2.2 Mackey-Glass Time Series Prediction Using FC Network

The Mackey-Glass (MG) equation is a nonlinear time delay differential equation originally developed for modeling white blood cell production [Mackey and Glass, 1977]. Its discrete time representation can be written as

$$x(t+1) = (1-B)x(t) + Ax(t-D)/\{1+x^{C}(t-D)\}$$

where A, B and C are constants and D is the time delay parameter. Under a suitable choice of these parameters, the resultant time series will exhibit chaotic behavior. The most extensively studied case [Casdagli, 1989; Mead et al., 1992; Azoff, 1994; Müller, Reinhardt and Strickland, 1995; Lin and Lee, 1996], with $A = 0.2$, $B = 0.1$ and $C = 10$, and the delay parameter D set to 30, is also selected here.

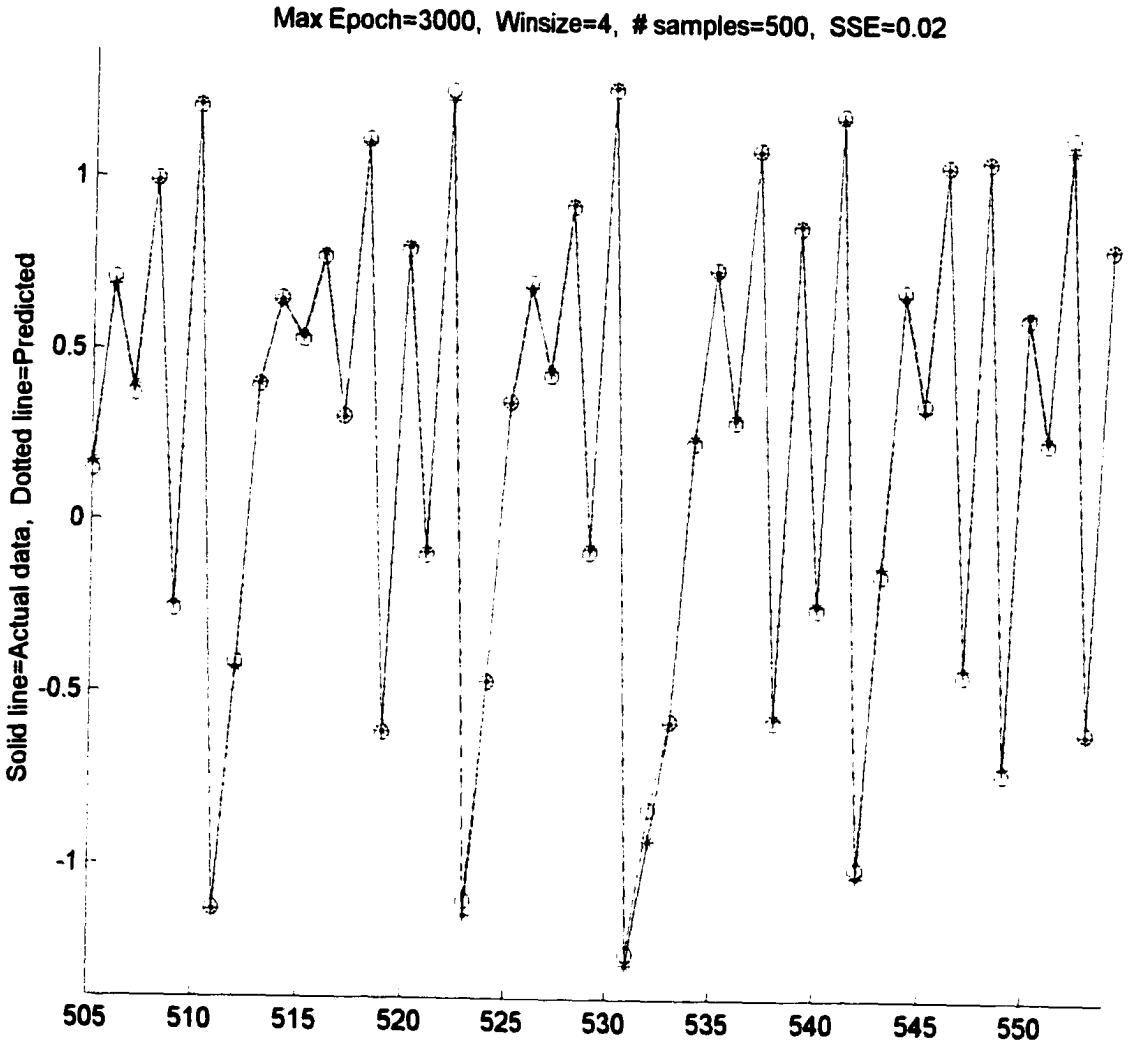

Fig. 20: Henon map time series prediction using BP (4-9-1)

In accordance with previously published work, the Mackey-Glass equation is used to generate a continuous sequence of data points, starting from the initial condition $x(t) = 0.9$ for the first 30 points. The first 3000 points are discarded to allow initialization transients to decay. The remaining data points are then sampled once every six points to obtain the actual time series used for this experiment. The window size is 6 and prediction is made for one point ahead. A total of 500 samples are used for training. The FC network architecture required to learn this time series is therefore 6-500-1. Since this time series is not as volatile as the Henon map time series, the number of points used for out-of-sample testing is increased to 100 without compromising the readability of the display and printouts. The result obtained for various values of k is shown in Table 5.

Table 5: Mackey-Glass time series prediction using FC network

k	SSE
1	0.3361
2	0.1999
3	0.1575
4	0.1461
5	0.1437
6	0.1569
7	0.1715

The best performance is obtained with $k = 5$. Figure 21 shows a plot of the predicted points against the actual data points for the out-of-sample testing phase. Again, the predicted points match the actual data points very well and most of the turning points in the time series have been correctly predicted. The sum-of-squared error (SSE) shown on top of the plot is the total prediction error cumulated over all 100 out-of-sample testing points. This plot is visually comparable to that obtained in a

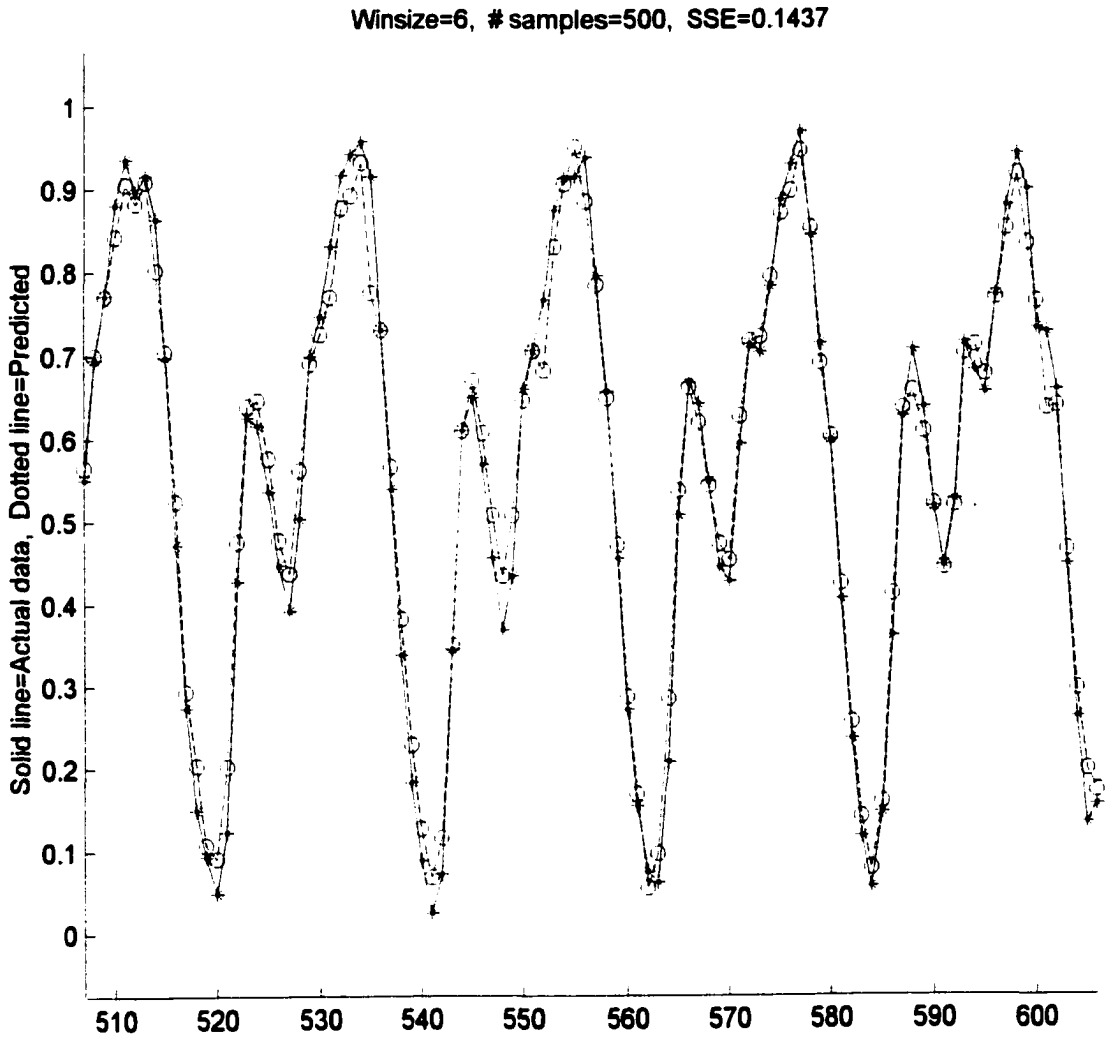

Fig. 21: Mackey-Glass time series prediction using FC (6-500-1), $k = 5$

similar experiment reported in the literature [Figure 6.2, page 100, Azoff, 1994] which was performed using a BP network implementing the Hertz rules adaptive steepest descent gradient method. A more thorough comparison with CC4, RBF and BP networks shall be conducted below.

(A) Comparison with CC4 Network

As for the Henon map experiment, a search is conducted to determine the optimum CC4 network parameters for this Mackey-Glass time series. Table 6 shows the result of the search.

Table 6: Mackey-Glass time series prediction using CC4 network

Qlevel (bits)	Best *r*	SSE
20	7	0.4950
30	10	0.3303
40	**14**	**0.2285**
50	18	0.3409
60	22	0.2997
70	29	0.3831
80	30	0.2625
90	32	0.2633
100	38	0.2773

The lowest SSE occurs for a quantization level of 40 and a radius of generalization of 14. With a window size of 6, the CC4 network architecture required for this time series is thus 241-500-40. Figure 22 shows the plot of the predicted points against the actual data points. Comparing it with the plot shown in Figure 21 for the FC network, it is clear that the FC network, with an SSE of 0.1437, again outperforms this CC4 network, which has an SSE of 0.2285.

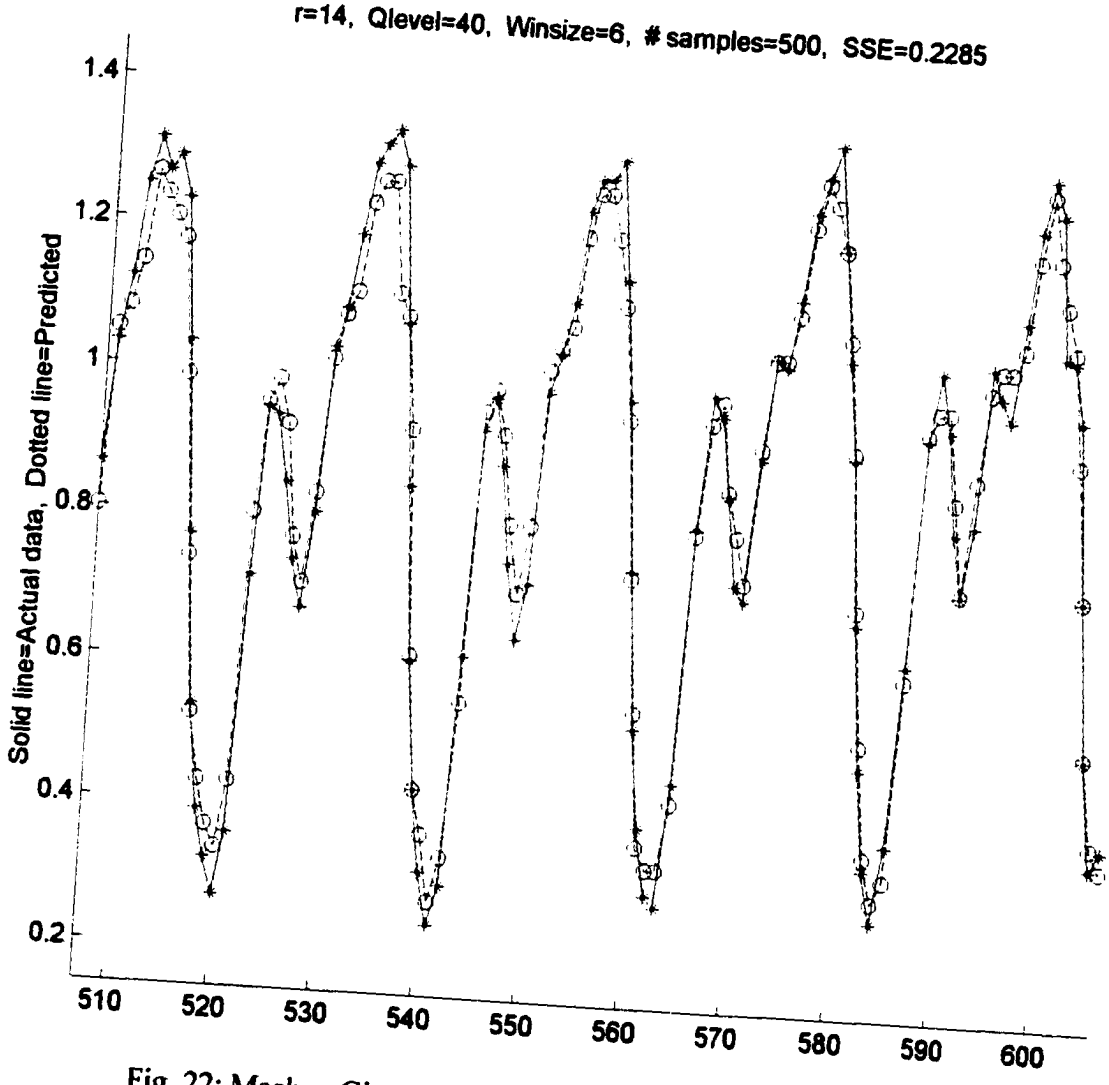

Fig. 22: Mackey-Glass time series prediction using CC4 (241-500-40)

(B) Comparison with RBF Network

The Mackey-Glass experiment is repeated using a RBF network designed and trained by the Matlab **solverbe** function. The network architecture needed to learn this time series is thus 6-500-1, which is the same as that for the FC network. As before, it is necessary to find the optimum spread constant for this RBF network. However, the search process is made difficult by the presence of numerous local minima, as shown in the Table 7, 8 and 9.

Table 7: Mackey-Glass time series prediction using RBF network

Spread Constant	SSE
0.10	10.4699
0.20	**0.1648**
0.30	0.2750
0.40	0.2689
0.50	0.2155
0.60	0.2887
0.70	0.3708
0.80	0.6092
0.90	0.5572

Table 8: Second-level search

Spread Constant	SSE	Spread Constant	SSE
0.11	11.3295	0.21	0.1749
0.12	13.6490	0.22	0.1877
0.13	4.9926	0.23	0.2025
0.14	5.8743	0.24	0.2189
0.15	7.0596	0.25	0.2367
0.16	8.5415	0.26	0.2555
0.17	10.3030	0.27	1.5004
0.18	**0.1564**	0.28	0.2702
0.19	0.1581	0.29	0.2727
0.20	0.1648		

Table 9: Third-level search

Spread Constant	SSE
0.180	0.1564
0.181	**0.1563**
0.182	0.2624
0.183	0.1563
0.184	0.1564
0.185	0.1566
0.186	0.1568
0.187	0.1570
0.188	0.1574
0.189	0.1577

The optimum spread constant is eventually found to be 0.181. Figure 23 shows the output obtained using this network. Comparing this plot with that shown in Figure 21, it is clear that the FC network outperforms this RBF network. As with the Henon map experiment, it is evident from Table 8 that the performance of this RBF network is again highly sensitive to changes in value of the spread constant. For example, a change in value of the spread constant from 0.18 to 0.17 causes an increase of SSE from 0.156 to 10.3.

(C) Comparison with BP Network

The Mackey-Glass experiment is repeated using a BP network trained by the Matlab **trainbpx** function. A similar search procedure described for the Henon map time series is conducted to find the optimum number of hidden neurons necessary for the BP network to appropriately model this time series. Table 10 shows part of the search result.

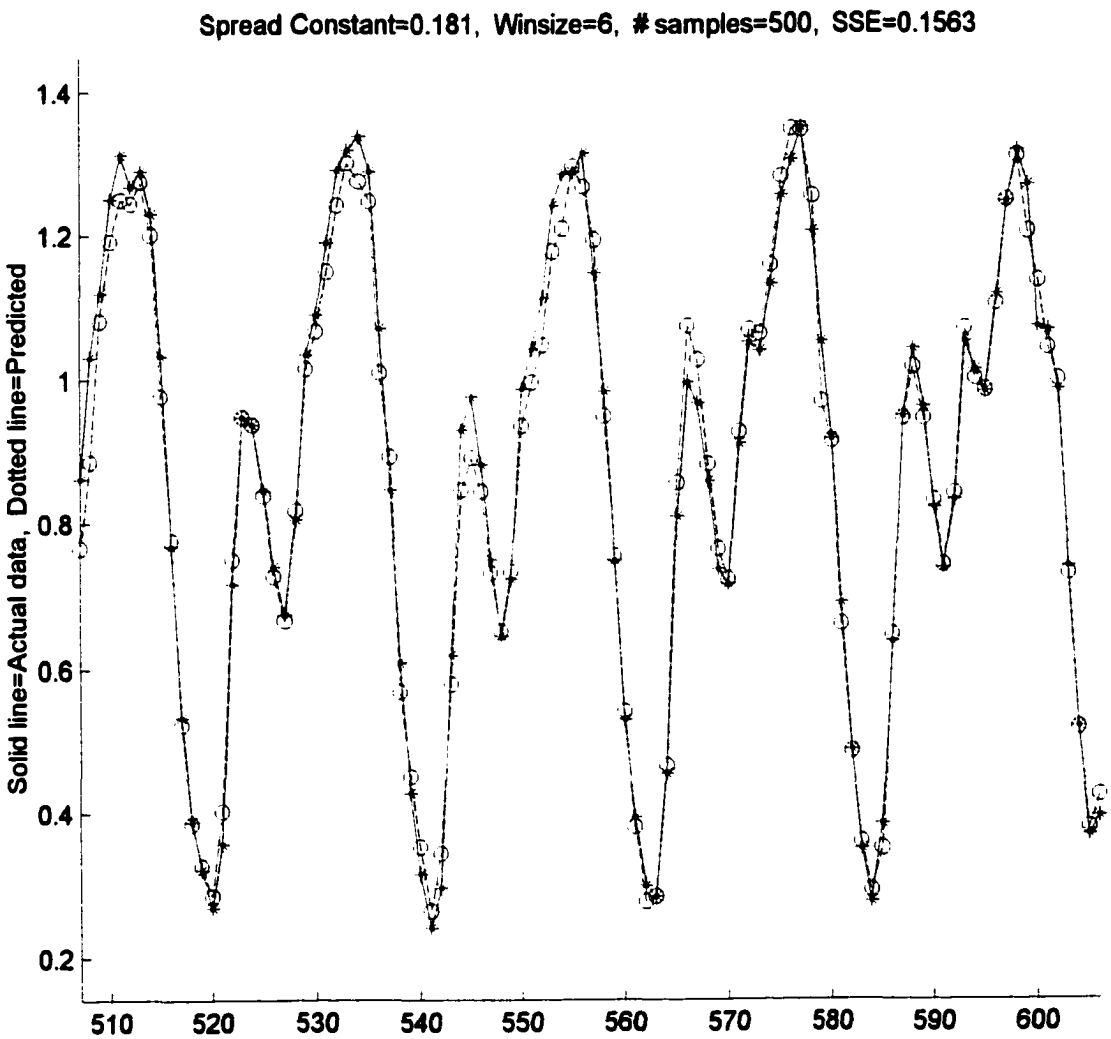

Fig. 23: Mackey-Glass time series prediction using RBF (6-500-1)

Table 10: Mackey-Glass time series prediction using BP network

Network Configuration	Average SSE over 10 runs
6-5-1	3.5289
6-6-1	2.3644
6-7-1	0.7698
6-8-1	0.7299
6-9-1	**0.3392**
6-10-1	2.3226
6-11-1	1.7222

Again, a network with nine hidden neurons seems to give the best performance. This network is then given more training. Figure 24 depicts a typical plot of the predictions made by this network after being trained for 5000 epochs. It shows that the performance of this network is marginally better than that of the FC network.

The results obtained so far in this chapter are summarized below for all four types of networks in order of performance.

Table 11: Time series prediction: summary of results

Network Type	Network Configuration	SSE
Henon map time series:		
RBF	4-500-1	0.00096
FC	4-500-1	0.00248
BP	4-9-1	0.0312
CC4	241-500-60	0.1462
Mackey-Glass time series:		
BP	6-9-1	0.1316
FC	6-500-1	0.144
RBF	6-500-1	0.1563
CC4	241-500-40	0.2285

This table shows that the performance of the FC network is comparable to that of RBF and BP networks and better than that of the CC4 network. However, it should

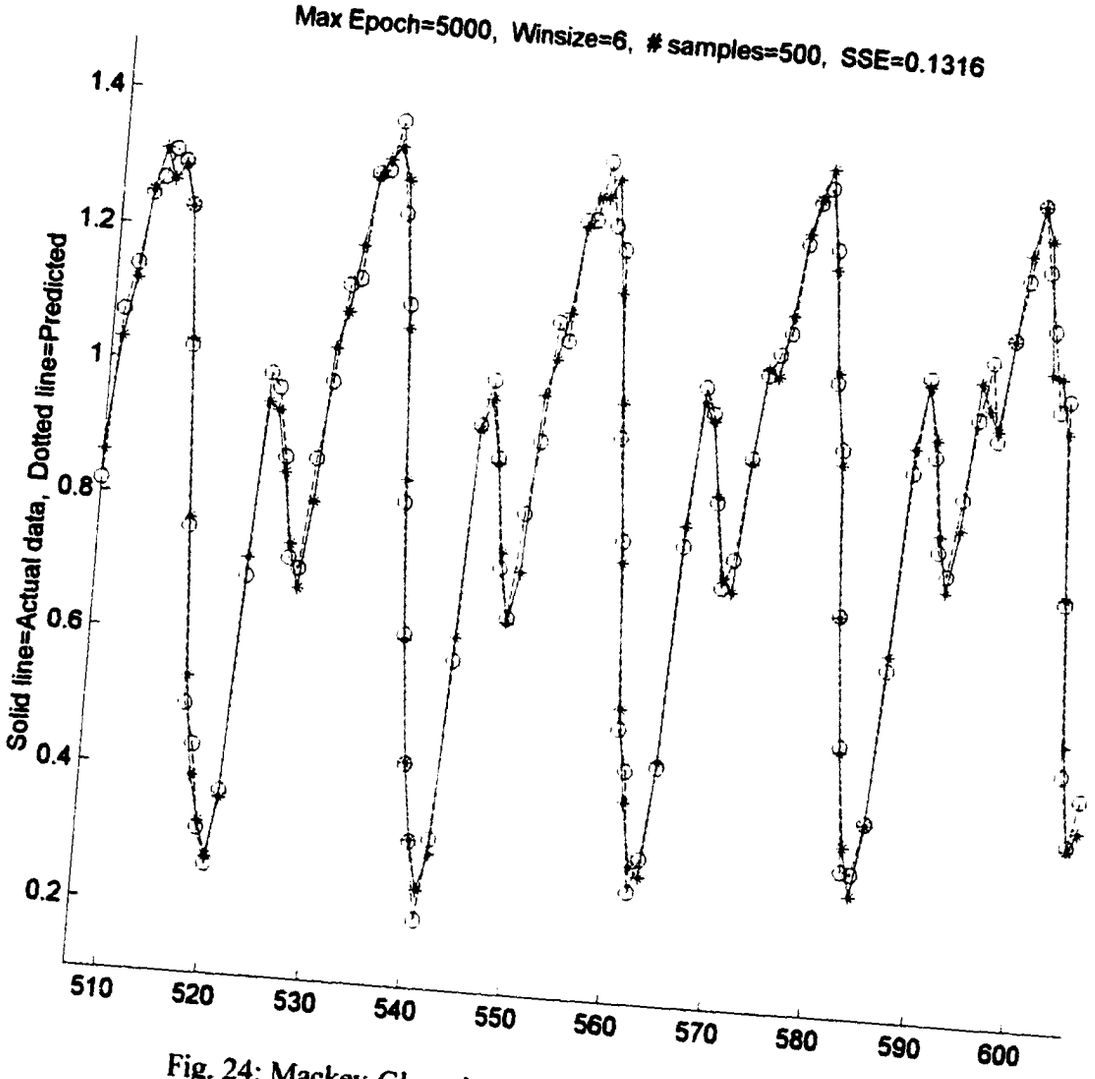

Fig. 24: Mackey-Glass time series prediction using BP (6-9-1)

be stressed that the comparison with the BP networks is by no means definitive. Given sufficient time, it is always possible to find a BP network that would solve a given problem to any desired degree of accuracy. However, the time needed to find such a BP network may be prohibitively long, particularly for large problems. Thus, most BP network users may just settle for a sub-optimal solution.

4.3 Performance Scalability

The window sizes used for experiments in this chapter so far have been chosen arbitrarily to be 4 for the Henon map and 6 for the Mackey-Glass time series. The various networks are then optimized for these window sizes. In a real-world application, it is often necessary for the user to try different window sizes in order to find one that is suitable for a given time series. It is thus of interest to find out how changing the window size would affect the performance of these optimized networks.

Further, in time series prediction, some historical data may be too old to be of any predictive value. Thus, the user must train the neural network with different amounts of historical data (i.e., different sample sizes) and then check how well the network performs in each case. It is thus desirable for a network that performs well at one sample size to perform consistently when a different sample size is used. This assures the user that if a substantial change in network performance is observed when a different sample size is used, it is likely to be due to the change in sample size itself rather than the inherent sensitivity of the network to variations in sample size.

In this section, the FC, CC4 and RBF networks shall be compared in terms of scalability of performance with respect to changes in window size as well as sample

size. The BP network is excluded in this exercise since the time required to find a suitably trained BP network for each case would be prohibitively long. Further, comparison with a BP network is not particularly meaningful for reasons already mentioned at the end of last section.

Toward this end, the Henon map and Mackey-Glass experiments are repeated with window sizes between 4 and 10 in step of 2, and sample sizes between 200 and 500 in step of 100. For each time series, the optimized parameters for each network, namely, k for the FC network, radius of generalization r and quantization level for the CC4 network, and the spread constant for the RBF network, are kept constant at their respective values found previously. Results for this scalability test are shown in the following tables.

Table 12: Henon map time series prediction: performance scalability

Window size	4	6	8	10
Sample size = 500				
SSE for FC	0.00248	0.01	0.07	0.17
SSE for CC4	0.1462	8.52	30.32	57.45
SSE for RBF	0.00096	0.40	1.84	9.10
Sample size = 400				
SSE for FC	0.0077	0.03	0.31	0.70
SSE for CC4	13.24	44.99	80.10	126.74
SSE for RBF	0.0002	0.60	8.15	23.54
Sample size = 300				
SSE for FC	0.017	0.054	0.28	0.73
SSE for CC4	13.30	52.42	93.54	119.45
SSE for RBF	0.23	4.01	9.51	34.41
Sample size = 200				
SSE for FC	0.035	0.13	0.44	0.97
SSE for CC4	7.49	44.85	88.94	147.98
SSE for RBF	0.041	3.27	8.35	73.52

Table 13: Mackey-Glass time series prediction: performance scalability

Window size	4	6	8	10
Sample size = 500				
SSE for FC	0.90	0.144	0.07	0.08
SSE for CC4	1.71	0.2285	2.21	4.10
SSE for RBF	532.77	0.1563	0.13	0.21
Sample size = 400				
SSE for FC	0.98	0.28	0.13	0.20
SSE for CC4	1.79	1.21	4.80	22.62
SSE for RBF	3510.5	4.28	0.19	0.44
Sample size = 300				
SSE for FC	1.42	0.50	0.34	0.33
SSE for CC4	1.70	2.46	33.28	50.05
SSE for RBF	33.65	3.39	1.04	2.55
Sample size = 200				
SSE for FC	1.88	1.20	0.92	1.17
SSE for CC4	2.56	9.63	35.48	51.24
SSE for RBF	443.55	4.25	4.49	6.08

From these tables, it can be seen that the performance of the FC network remains good and reasonably consistent throughout all window sizes and sample sizes while that of the CC4 and RBF networks are adversely affected by changes in the window size or the sample size or both. Indeed, the performance of the CC4 and RBF networks can become erratic at certain combinations of these parameters. This implies that an FC network designed for one window size or one sample size is generally applicable to other window sizes and sample sizes. On the other hand, each time the window size or sample size changes, a new CC4 or RBF network may be required. It is therefore clear that the FC network is very much easier to use in practice compared to the other networks.

Table 13 also reveals another interesting feature in that a value of 6 is not the best window size to use for the FC network in modeling this Mackey-Glass time series. A window size of 8 gives better performance for all sample sizes tested. This is probably due to the low volatility in this time series which is better modeled by a longer window size.

4.4 Summary

This chapter tests the performance of the FC network in the context of time series prediction using the highly volatile Henon map time series and the less volatile Mackey-Glass time series. In both cases, the value of k that gives the best performance for a sample size of 500 is 5. Thus, k is just 1% of the sample size, which is consistent with the discussion at the end of the last chapter.

The FC network is then compared with the CC4, RBF and BP networks. It is found that the generalization performance of the FC network is comparable to that of the RBF and BP networks and better than that of the CC4 network. Further, the scalability of the performance of the FC network with respect to changes in window size and sample size is found to be superior compared to that of the CC4 and RBF networks. This property makes the FC network easy to use in practice, since an FC network that works well for one window size or sample size would also work well for other values of window and sample sizes.

Chapter 5

Pattern Classification Using Fuzzy Classification Neural Networks

Humans are good at pattern recognition. We can recognize the faces of friends, even after they have aged or put on considerable amount of weight since we last met them. We can identify a familiar person just by listening to his or her voice over the telephone, even though the connection is noisy. Likewise, we can differentiate one type of food from another just by smelling or tasting it. Although artificial neural networks are not nearly as good at solving similar problems, they attempt to use a similar processing style. With the help of suitable transducers, observations about the real world can be fed to the network. The network can store these information and it can also establish correspondence between past and present observations. It thus possesses the fundamental constituents of a basic pattern recognition scheme.

In this dissertation, the term *pattern classification* shall be used to refer to the process by which a network first learns the mapping between input patterns and output classes from a set of training samples and then attempts to assign new input patterns to one of the output classes predefined by the training samples. It thus includes the tasks of recognition, identification and classification.

This chapter discusses two methods of implementing a pattern classifier using the FC network, as described in Chapter 3. In the first method, a test pattern is

assigned fuzzy memberships of output classes corresponding to its k nearest neighbors. This method is called *classification by fuzzy membership*. In the second method, referred to as *classification by voting*, the notion of fuzzy membership is removed and the k nearest neighbors are allowed to vote. The test pattern is assigned to the class that gets the largest number of votes.

The implementation of each method shall be illustrated using a spiral pattern. Figure 25 shows a black-and-white spiral pattern within a 32-by-32 rectangular area. Approximately half the points within the rectangular area belongs to the black region while the remaining half belongs to the white region of the spiral. One-fourth of the total number of points are randomly selected from the black and the white region in roughly equal proportions. These selected points are shown in Figure 26. They are used as samples for training both the FC networks.

Input to the networks consists of the row and column coordinates of the training samples which are integers ranging from 1 to 32. The networks have two output neurons, one for each class. If a training sample belongs to the white region, the corresponding target output is the two-bit binary vector (1, 0). On the other hand, if the training sample belongs to the black region, its target output is the vector (0, 1). The FC network architecture required is therefore 2-256-2.

After training, all 1024 points within the rectangular area are presented as test vectors one at a time and the networks are required to classify each of the points into either the black or the white region. A point is classified to the region whose output neuron produces the highest output. The number of incorrectly classified points is noted for different values of k.

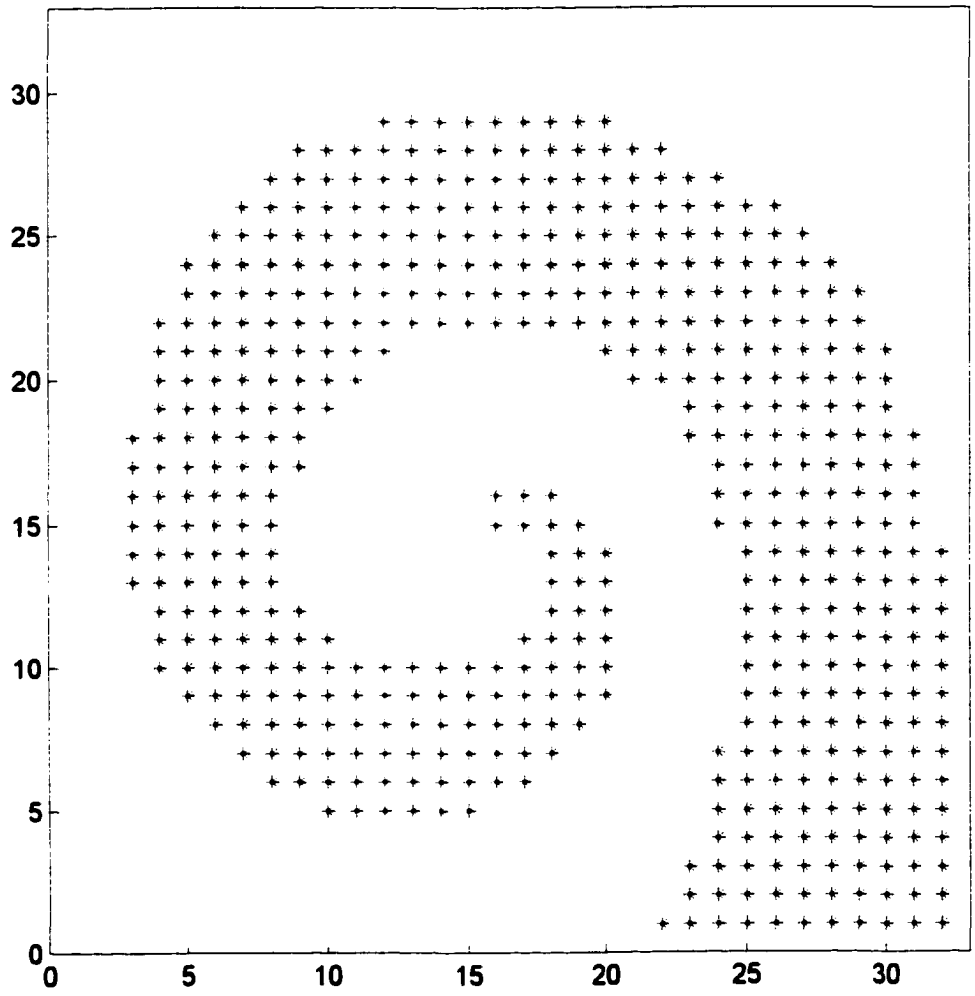

Fig. 25: 32x32 spiral

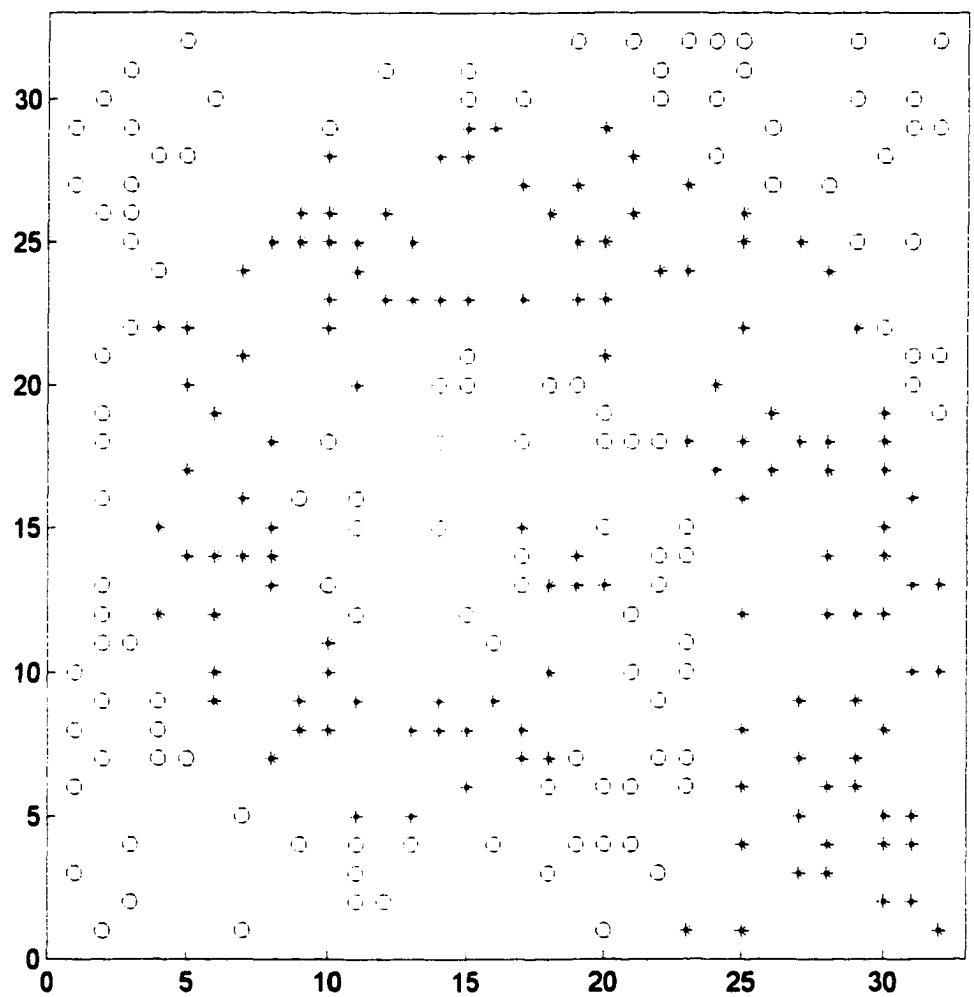

Fig. 26: 256 randomly selected samples

5.1 Classification by Fuzzy Membership

For this pattern classifier, the FC network is implemented according to Section 3.5. If an input vector results in a tie at the network output, it is assigned to the white region for simplicity. Table 14 shows the result.

Table 14: Spiral pattern classification by fuzzy membership

k	Classification Errors
1	58
2	61
3	55
4	52
5	47
6	50
7	49

This table shows that the best result is obtained when $k = 5$. Figure 27 shows the predicted spiral produced by this network. It can be seen that the predicted spiral resembles the original spiral very well with only 4.6% of the points misclassified.

5.2 Classification by Voting

For this pattern classifier, the FC network is implemented according to Section 3.6. Since the spiral pattern used here has only two output classes, a tie can be avoided by choosing k to be odd. The performance of this classifier is shown in Table 15.

Table 15: Spiral pattern classification by voting

k	Classification Errors
1	58
3	53
5	51
7	58

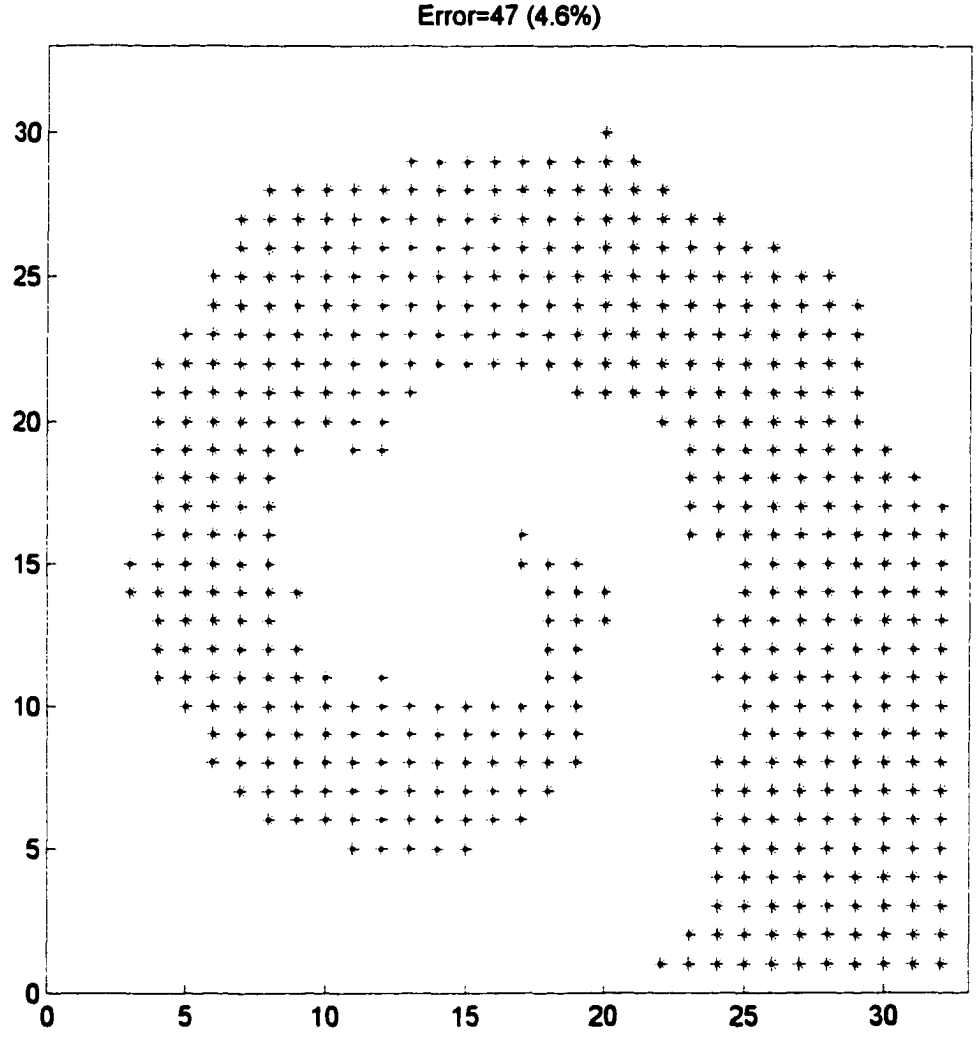

Fig. 27: Spiral pattern classification by fuzzy membership, $k = 5$

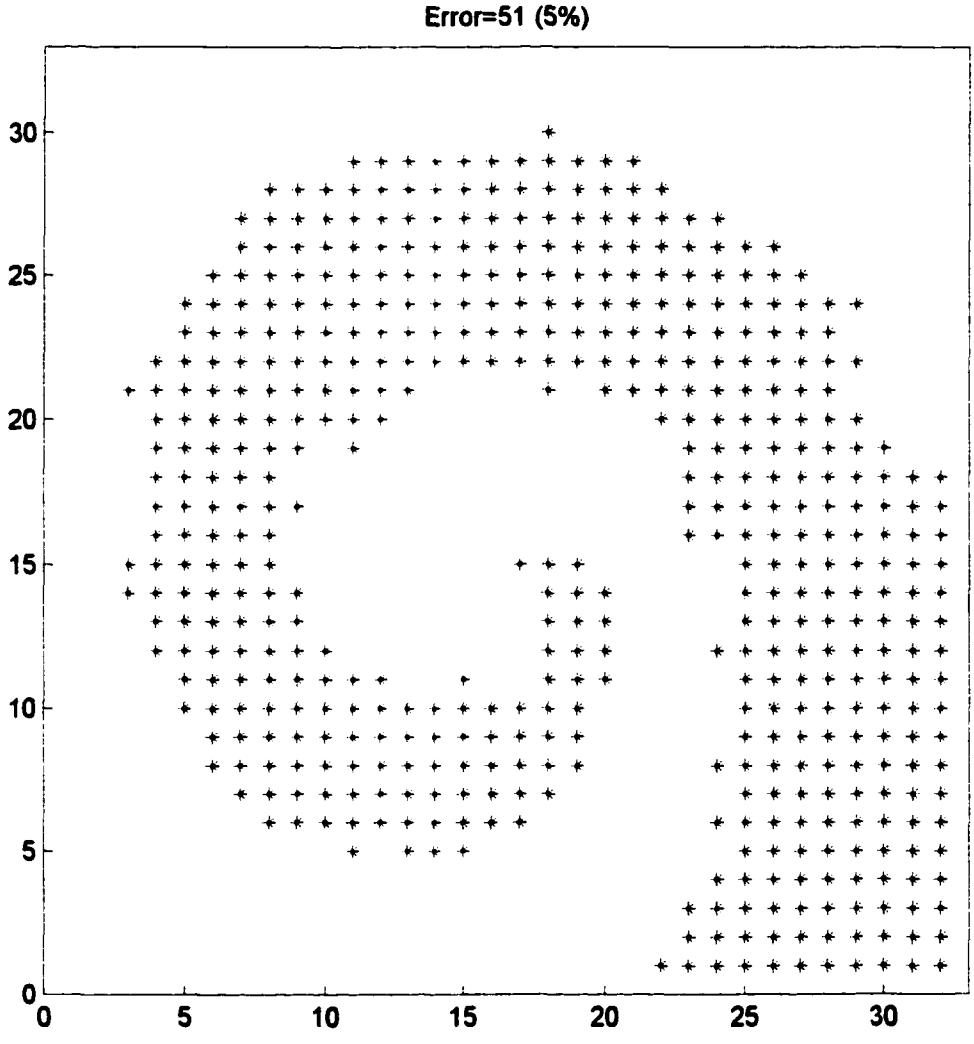

Fig. 28: Spiral pattern classification by voting, $k = 5$

It can be seen that the error rates for this classifier is generally higher than that of the previous one. When $k = 1$, the two networks become identical, both reducing to the single-nearest-neighbor classifier. Again, the best performance is obtained when $k = 5$, which gives an error of 51 or 5.1%. Figure 28 shows the predicted spiral.

5.3 Comparison between FC and Other Networks

The spiral pattern classification experiment is repeated here for the CC4, RBF and BP networks using the same set of training samples as that used by the FC network.

(A) Comparison with CC4 Network

Since the row and column coordinates of the points in the spiral pattern are integers ranging from 1 to 32, each coordinate can be encoded as a 32-bit unary vector for inputting to the CC4 network. As for the FC networks, two output neurons are required to encode the two output classes. The CC4 network architecture needed to learn this spiral pattern is therefore 65-256-2. The network is trained using various radii of generalization r, and its performance checked by counting the number of points wrongly classified. Table 16 shows the result of this experiment.

Table 16: Spiral pattern classification using CC4 network

r	Error
1	141
2	**78**
3	86
4	98
5	103

This table shows that the minimum prediction error occurs at $r = 2$. Figure 29 shows the spiral predicted by this CC4 network, where 7.6% of the points have been

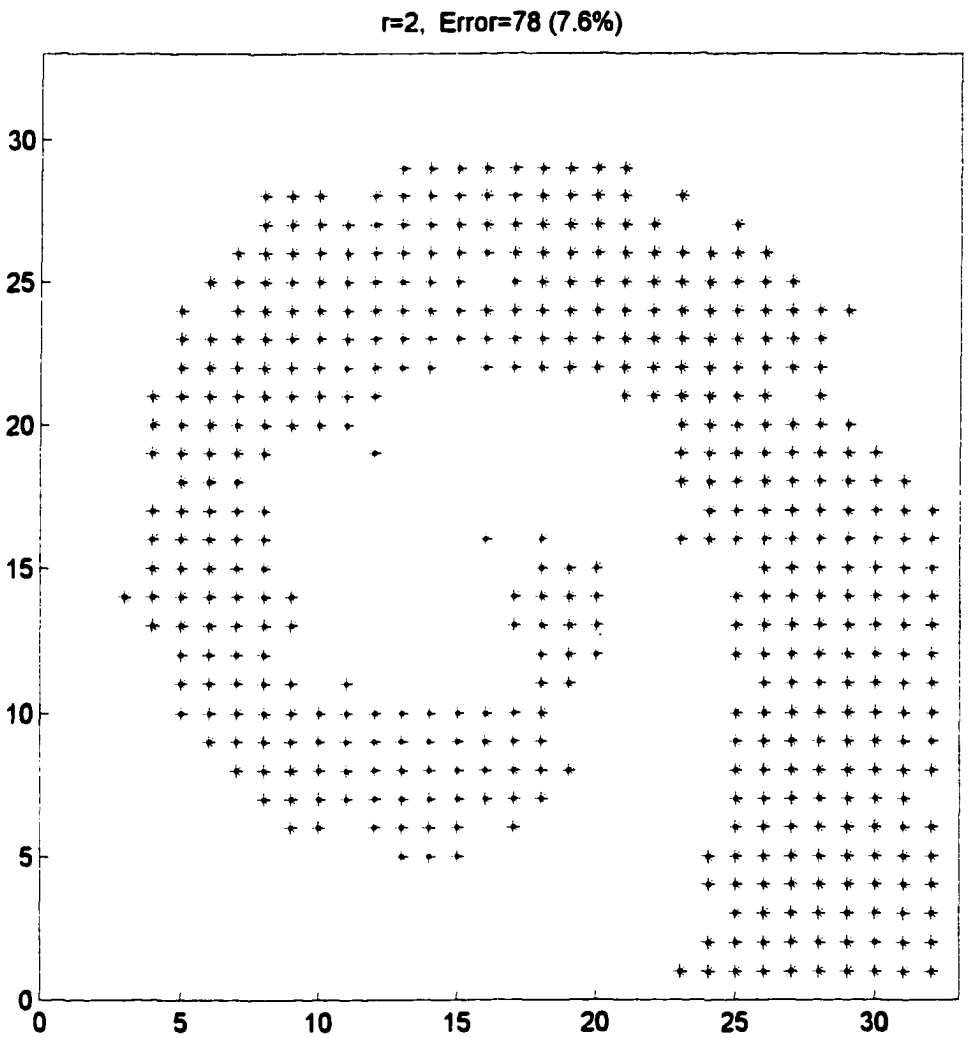

Fig. 29: Predicted spiral using CC4 (65-256-2)

misclassified. Comparing this with the two FC networks where the error rates are 4.6% and 5% respectively, it is clear that both versions of the FC network outperform the CC4 network by a wide margin.

(B) Comparison with RBF Network

The spiral pattern classification experiment is repeated here using a RBF network designed and trained by the **solverbe** function. As mentioned in the previous chapter, this version of the RBF network uses one hidden neuron to learn each training sample. The network architecture needed is 2-256-2, which is the same as that for the FC networks. Like the FC network, this RBF network is capable of producing zero network error on the training set. As for the time series experiments, a search is conducted to determine the optimum spread constant for the network for this pattern classification experiment. Result of the search is shown in Table 17, 18 and 19 respectively.

Table 17: Spiral pattern classification using RBF - initial search

Spread Constant	Error	Spread Constant	Error
1.0	159	2.0	79
1.1	112	2.1	87
1.2	86	2.2	90
1.3	79	2.3	97
1.4	69	2.4	105
1.5	91	2.5	112
1.6	85	2.6	124
1.7	82	2.7	134
1.8	100	2.8	143
1.9	111	2.9	146

Table 18: Spiral pattern classification using RBF - second search

Spread Constant	Error	Spread Constant	Error
1.30	79	1.40	69
1.31	78	1.41	68
1.32	77	1.42	67
1.33	77	1.43	99
1.34	76	1.44	95
1.35	75	1.45	92
1.36	72	1.46	62
1.37	71	1.47	90
1.38	71	1.48	89
1.39	70	1.49	88

Table 19: Spiral pattern classification using RBF - third search

Spread Constant	Error	Spread Constant	Error
1.450	92	1.460	62
1.451	92	1.461	62
1.452	92	1.462	62
1.453	92	1.463	62
1.454	92	1.464	62
1.455	92	1.465	62
1.456	63	1.466	63
1.457	63	1.467	63
1.458	63	1.468	63
1.459	63	1.469	63

As shown in Table 19, the optimum spread constant can be taken as 1.460.

Figure 30 shows the output obtained from this RBF network, which has an error rate of 6.1%. Comparing this result with that obtained by the FC networks, it is clear that both versions of the FC network perform better.

(C) Comparison with BP Network

The spiral pattern classification experiment is used here to compare the performance of BP networks trained by the **trainbpx** function. The same procedure as in the time

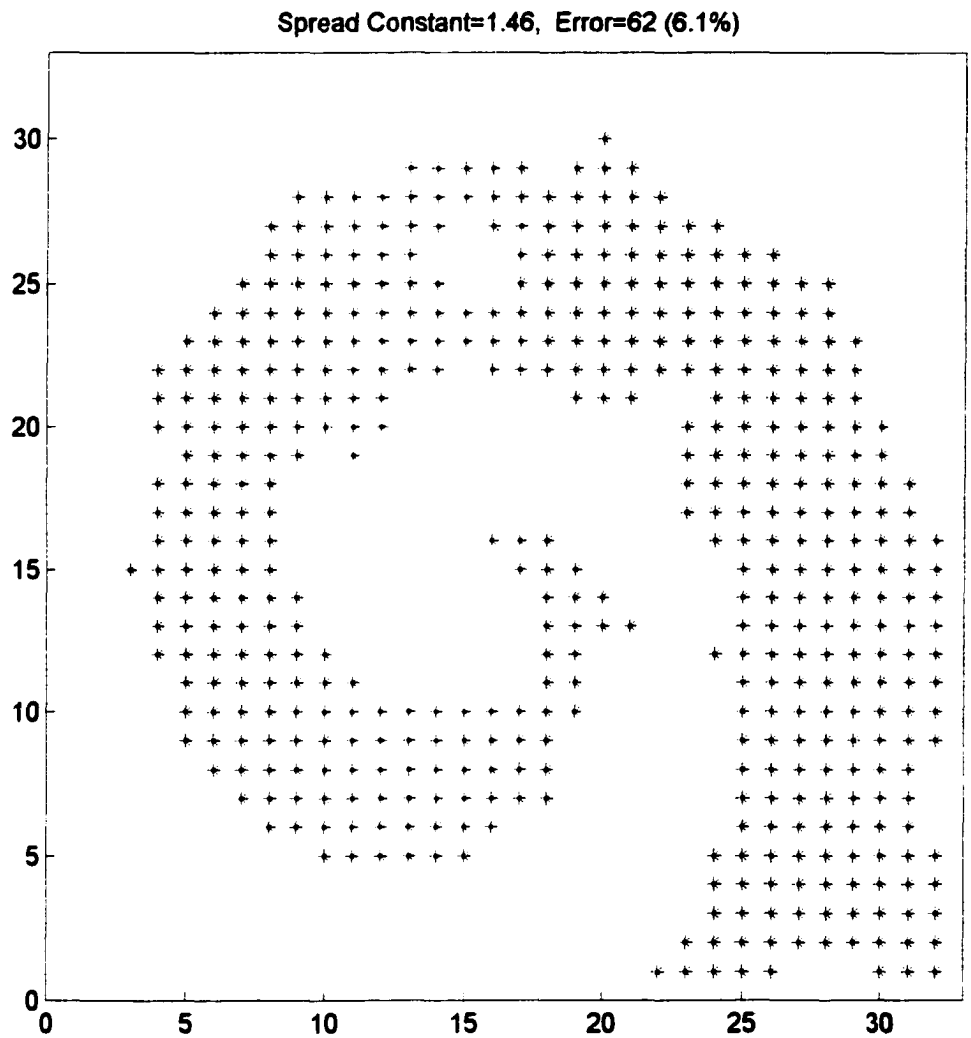

Fig. 30: Predicted spiral using RBF (2-256-2)

series prediction experiments is used here to find a suitable network configuration for learning the spiral pattern. Part of the result of the search is given in Table 20.

Table 20: Spiral pattern classification using BP network

Hidden Neurons	Average Error
20	268.3
25	237.8
30	172.3
35	211.3
40	161.5
45	115.9
50	92.2
55	101.4
60	71.8
65	133.8
70	95.3
75	93.0
80	148.0

This table shows that a network with 60 hidden neurons is capable of giving good performance. This network is then given more training. Figure 31 shows a typical predicted spiral produced by this network after being trained for 5000 epochs. It can be seen that the performance of this network is comparable to that of the FC networks.

The results obtained so far in this chapter are summarized in the following table in order of performance.

Table 21: Two-class spiral pattern classification - summary of results

Network Type	Network Configuration	Error
FC (Fuzzy Membership)	2-256-2	47
BP	2-60-2	48
FC (Voting)	2-256-2	51
RBF	2-256-2	62
CC4	65-256-2	78

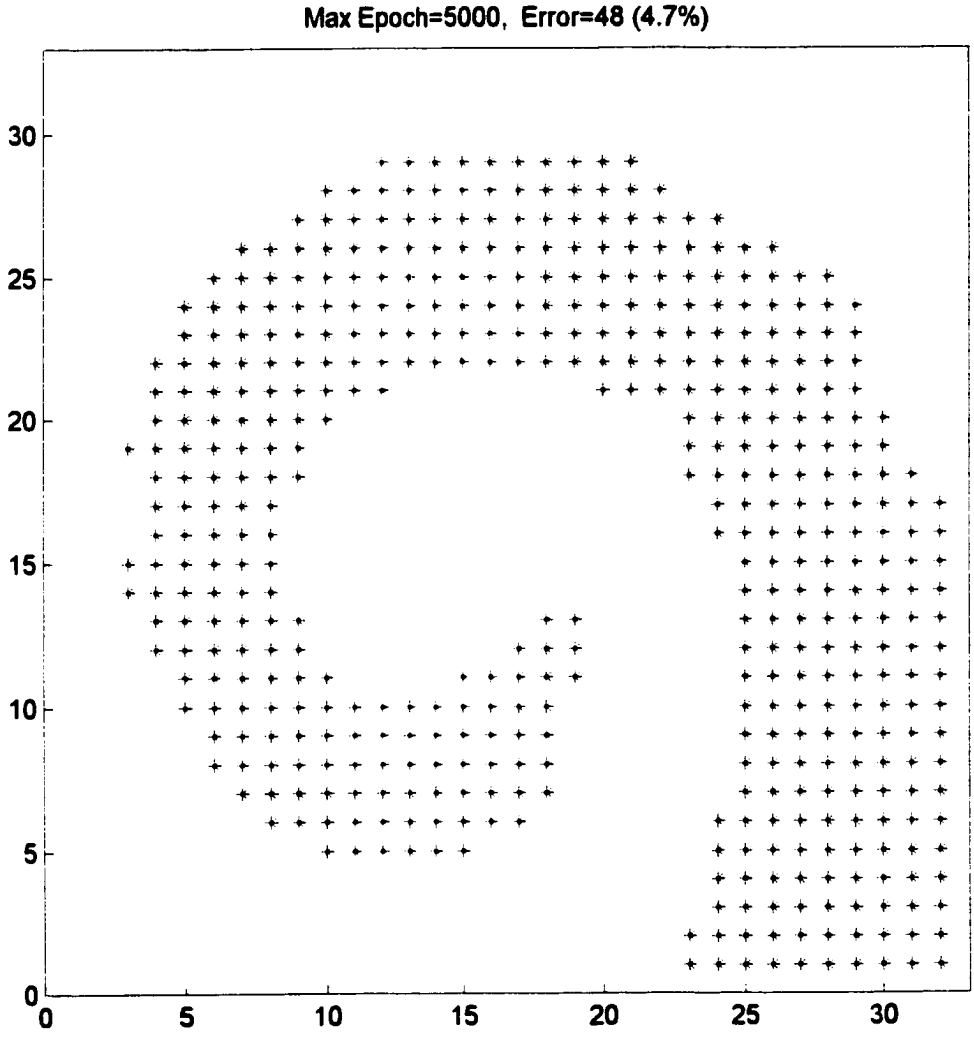

Fig. 31: Predicted spiral using BP (2-60-2)

5.4 Performance Scalability

The spiral pattern used in experiments so far consists of only two regions; i.e., the problem involves only binary decision. However, in many real-world pattern classification applications, it is common to find problems involving more than two output classes. Further, having trained a neural network based on a certain number of output classes initially, it is often necessary to increase or decrease the number of classes due to changes in specifications of the problem at hand. Thus, it is desirable that a neural network that works well initially should not deteriorate substantially in performance when the number of output classes changes.

In this regard, the performance of the FC, CC4 and RBF networks shall be tested when the input patterns in the spiral must be classified into four classes instead of just two. Only the FC network with classification by fuzzy membership shall be used, since it is the intended mode of operation and it gives better performance than classification by voting. The BP network is not included in this exercise for the reasons that it is too time-consuming while not providing meaningful comparison.

The spiral pattern for this four-class problem is shown in Figure 32 while the training samples are shown in Figure 33. Comparing Figure 33 and Figure 26 for the spiral with two regions, it can be seen that the position of the training samples (i.e., the input vectors) remains unchanged. What has changed in Figure 33 is the target output for each sample (i.e., the output classes). These samples are used to train the FC, CC4 and RBF networks used earlier for the two-category problem. After training, the networks are used to classify each of the 1024 points in the spiral pattern, and the

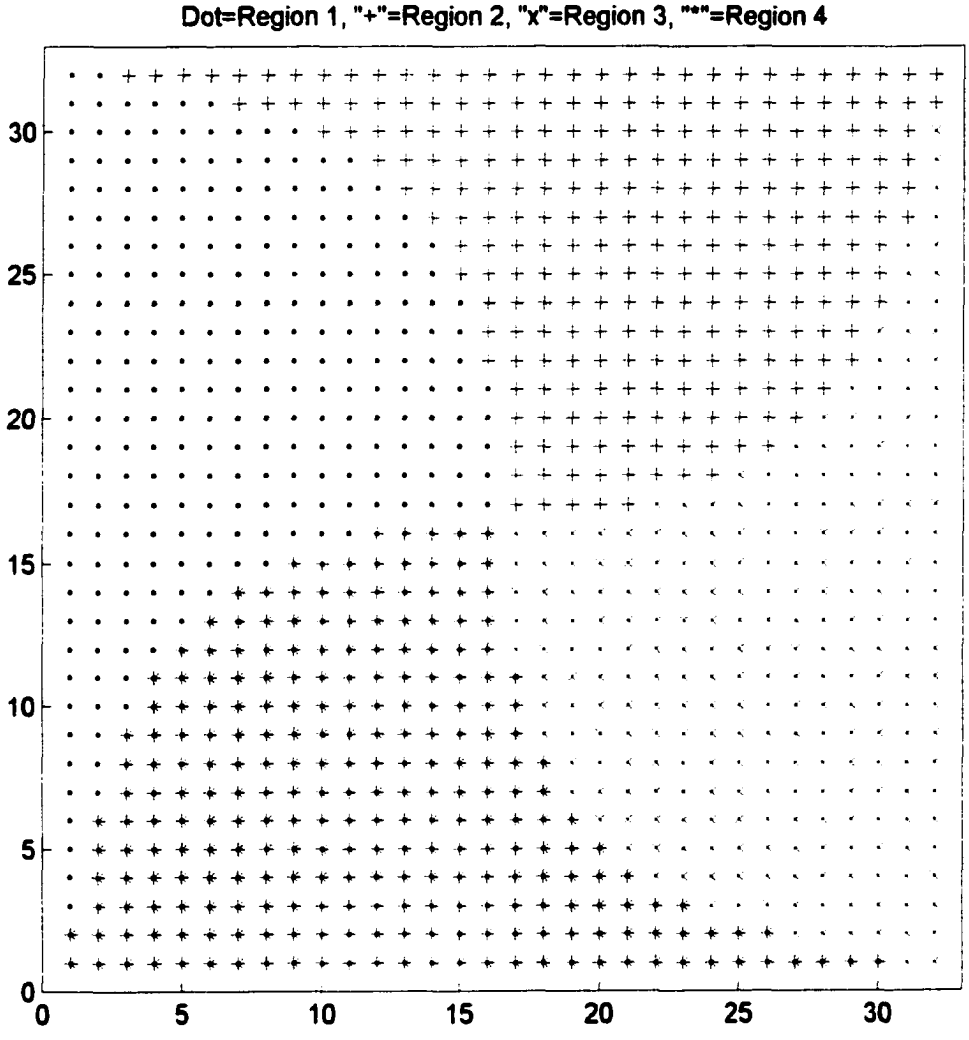

Fig. 32: 32x32 spiral with 4 regions

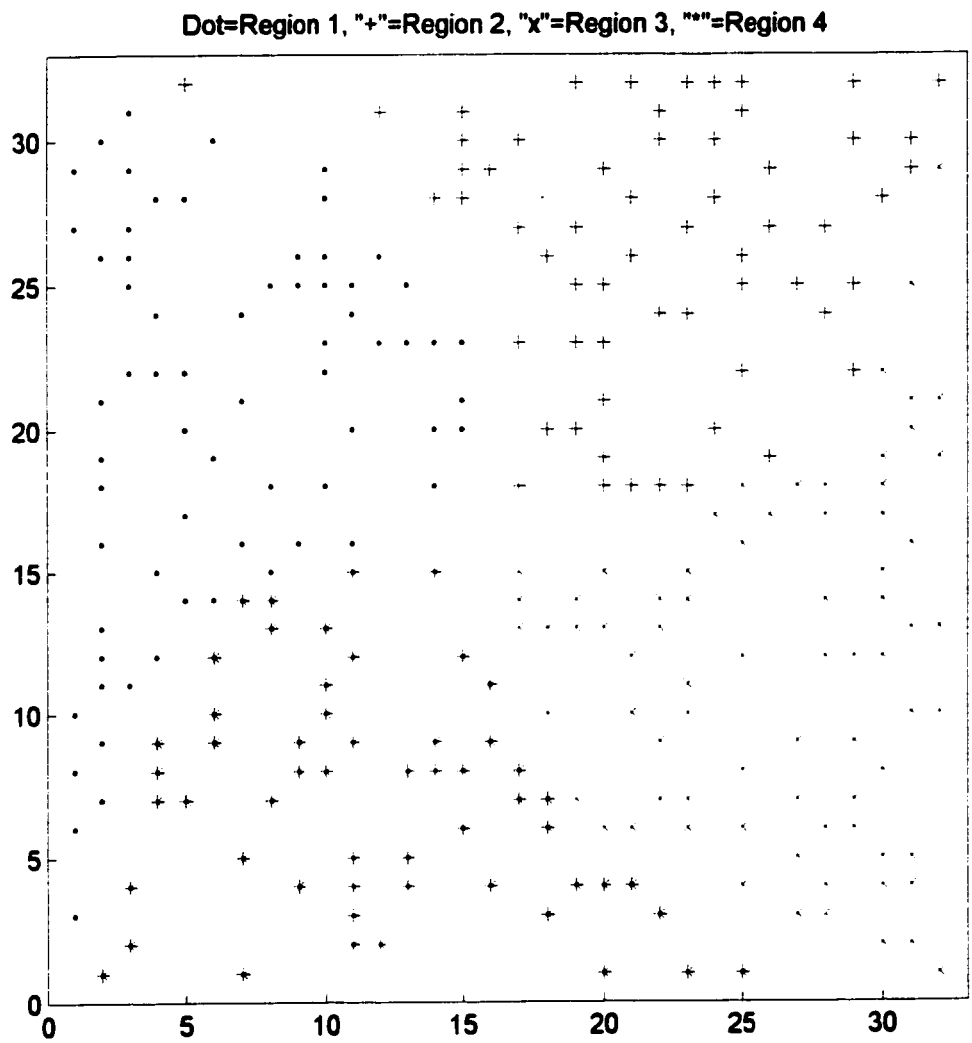

Fig. 33: 256 randomly selected samples

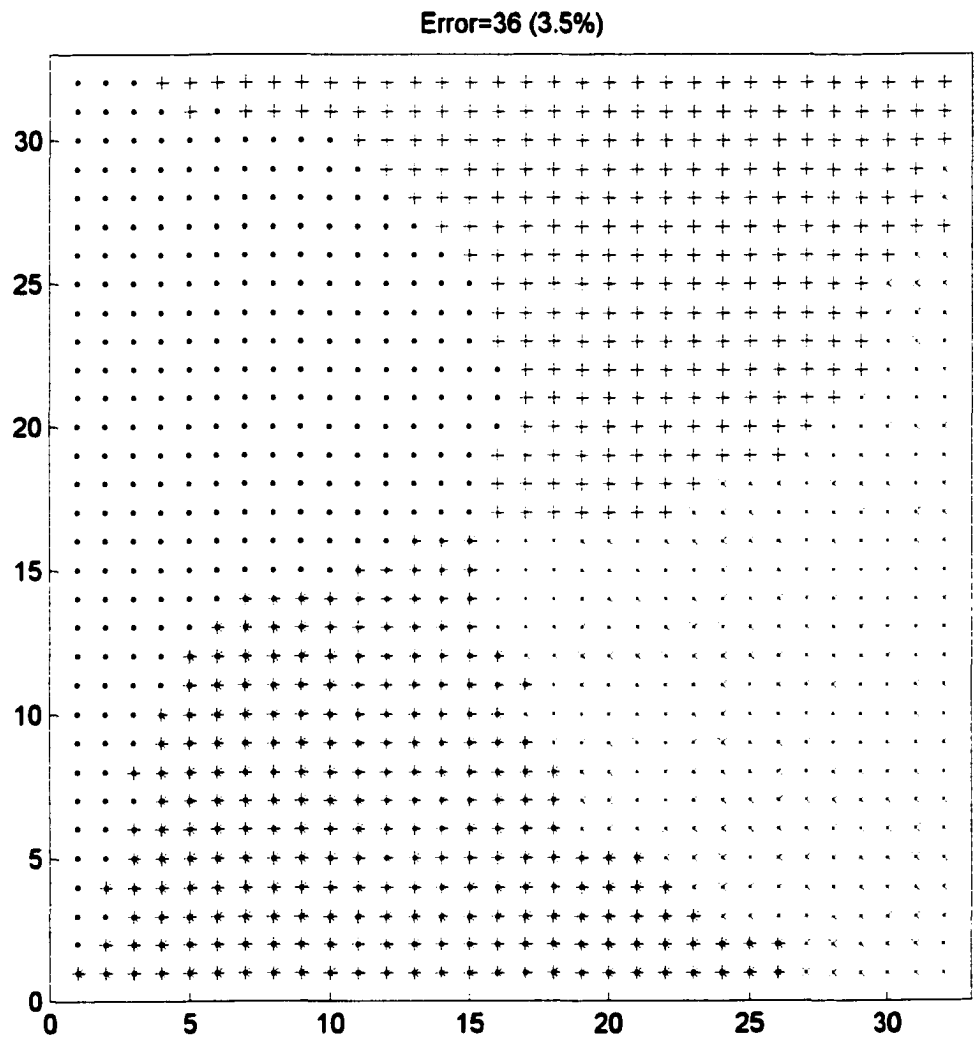

Fig. 34: Four-class spiral predicted by FC (2-256-4), $k = 5$

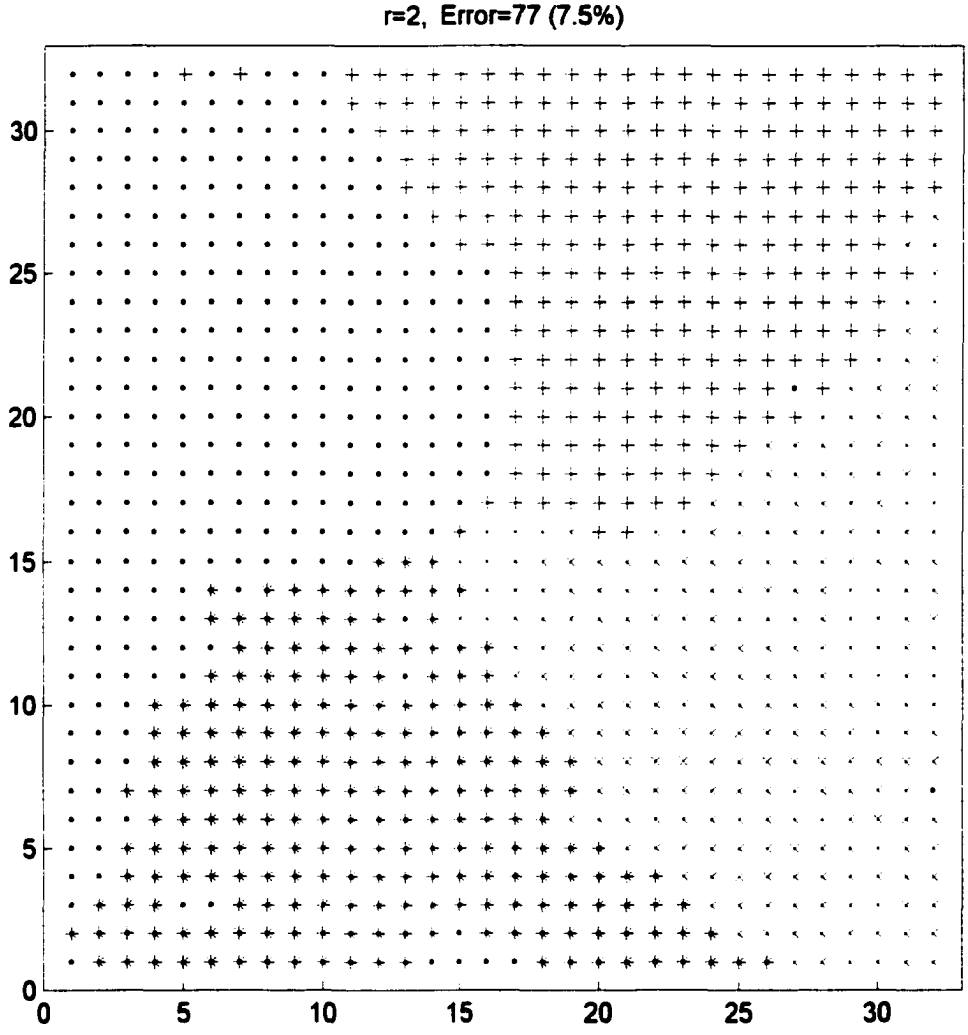

Fig. 35: Predicted spiral using CC4 (65-256-4)

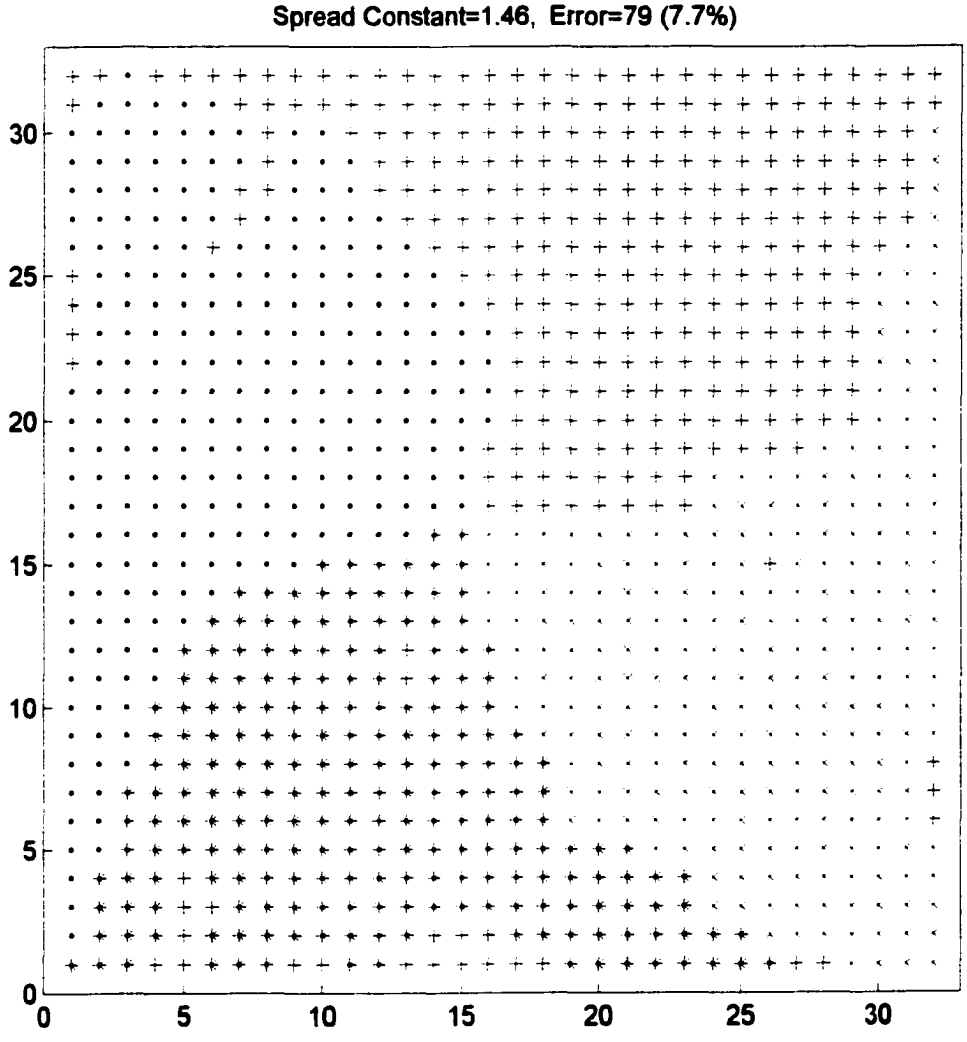

Fig. 36: Predicted spiral using RBF (2-256-4)

number of misclassified points are tallied. The predicted spiral from these three networks are shown in Figure 34, 35 and 36 respectively while the number of misclassified points is shown in Table 22. It is clear that the FC network again outperforms the CC4 and RBF networks by a wide margin. Further, comparing the number of misclassified points in Table 21 and 22, it is seen that the FC and CC4 networks scale very well with respect to the number of output classes while the RBF network is adversely affected. This implies that a new RBF network with a different spread constant should be used when the number of output classes changes.

Table 22: Four-class spiral pattern classification - summary of results

Network Type	Network Configuration	Error
FC	2-256-4	36
RBF	2-256-4	77
CC4	65-256-4	79

In all experiments in this chapter, the number of training samples used is 256, which is one fourth of the total number of points that make up the spiral. In a real-world situation, collecting training data is often time-consuming and costly. The user may therefore be forced to work with a limited set of training data. To see how reducing the size of the training set affects the performance of the FC, CC4 and RBF networks, both the two-class and four-class spiral problems are repeated using reduced sample sizes of 200 and 150. The reduction in sample size is done in such a way as to ensure that the remaining training samples are evenly distributed in the input space as far as possible. The following table summarizes the results on this performance scalability test.

Table 23: Performance scalability - summary of results

Sample size	256	200	150
Two-class spiral pattern:			
Error for FC	47	46	51
Error for CC4	78	89	107
Error for RBF	62	60	310
Four-class spiral pattern:			
Error for FC	36	46	54
Error for CC4	77	93	126
Error for RBF	79	91	133

It can be seen that the performance of the FC network scales very well with respect to the number of output classes as well as the size of the training set. On the other hand, while the CC4 network scales fairly well in terms of changes in number of output classes, it performs poorly when the sample size is reduced. Finally, the RBF network does not scale well at all. This implies that each time the number of output classes or the number of training samples changes, a new RBF network must be found.

5.5 Summary

This chapter tests the performance of the FC network as a pattern classifier using a spiral pattern. Two versions of the FC network, namely, classification by fuzzy membership and classification by voting, are tested. Both versions give the best performance when $k = 5$, which is a small fraction of the sample size. This is consistent with the result obtained in the previous chapter as well as the discussion at end of Chapter 3. The FC networks are also compared against the CC4, RBF and BP networks. It is found that the FC network with classification by fuzzy membership gives the best performance of the five networks. This version of the FC network is

further compared with the CC4 and RBF networks in the performance scalability test. This test shows that the performance of the FC network as a pattern classifier is consistent and good with respect to changes in the number of output classes as well as reduction in sample size.

Chapter 6

Conclusion

This dissertation presents the new Fuzzy Classification Neural Network. It has the excellent capability of instantaneous learning and its performance for both time series prediction and pattern classification has been shown to be good. In particular, its generalization ability compares very favorably with that of mainstream neural networks such as the BP networks and the RBF networks. Further, the FC network has consistently outperformed the CC4 network in all experiments. An enhanced version of the CC4 network which adaptively assigns appropriate radius of generalization to hidden neurons has since been developed [Souza and Kak, 1998]. It has been shown to improve performance substantially. However, it does not have the instantaneous learning capability of the original CC4 network.

From the various experiments performed in this dissertation, it is clear that designing an FC network for any given real-world problem is easy compared to the CC4, RBF or BP networks. Its performance is also more scalable. In conjunction with its fast training speed, the FC network therefore offers an attractive alternative to the other networks.

Although several types of neural networks with instantaneous learning capability are mentioned in Chapter 2, only the CC4 network has been used for

performance comparison in this dissertation while the remaining networks have not. This is because of the following reasons:

1. The WISARD network is strictly a pattern classification network. It is not suitable for function approximation application such as time series prediction. It is intended for implementation in hardware using Random Access Memory (RAM) chips whereas the other types of networks discussed in this dissertation are all implemented in software.
2. The PNN is also strictly a pattern classification network. It is not used for comparison in this dissertation because it is felt that the performance of the PNN network can be easily influenced by some subjective choice of network parameters such as h_k and l_k, thus resulting in misleading comparison. Likewise, the GRNN, which is based on the PNN, is strictly a function approximation network and not suitable for use in pattern classification.

A very strong advantage of the FC network compared to other networks such as the BP network is that it can be used for real-time applications in which it is necessary to retrain the network as soon as a new data point becomes available. Retraining the FC network involves simply adding a hidden neuron to learn the new data point and repeating step 2 of the training process to update the radius of generalization for the hidden neurons to account for the new data point. By contrast, retraining a BP network is not much different from training the original network and thus, is subject to the same set of problems as training the original network. Further, for time-varying data set, the FC network can overwrite old training samples with new ones to keep the network up to date always. As soon as the FC network has learned one sample representing a new

output class, it can immediately begin to generalize using that new sample. As more samples belonging to that output class become available, the generalization would improve.

A major disadvantage of the FC network, as with all neural networks with instantaneous learning capability, is that it requires one hidden neuron for each training sample. This can give rise to two problems, particularly when the training set is large:

1. The memory required to store the training samples can be large.
2. The time taken by the FC network to compute an output can be slightly longer than that of a more compact network such as a BP network.

However, the first problem is becoming less severe since the FC network is meant to be implemented in a software program and the current breed of low-cost desktop computers have very large memory capacity, typically measured in terms of megabytes. Similarly, the second problem is partially mitigated by the current breed of high-power microprocessors with clock speed approaching the GHz range. For extremely large data set, these problems can be overcome by preprocessing the data using one of various types of clustering algorithms available.

Although modeling biological learning is not a main objective of this dissertation, it is worthwhile mentioning that instantaneous learning of the kind achieved by the FC network could be a plausible mechanism for biological learning, especially in *working memory*. The notion of working memory was recently proposed by neuroscientists [Fuster, 1995; Daneman and Merikle, 1996; D'Esposito and Grossman, 1996; Goldman-Rakic, 1996; Jonides et al., 1996]. Working memory operates over a time span of seconds and it appears to be a central element in the

organization of behavior, language, and thinking [Wickelgren, 1997]. It is supposed to briefly store and process information for planning and reasoning [Baddeley and Sala, 1996]. Such a short-term memory requires very quick learning, which can be modeled very well using the instantaneous learning mechanism in the FC network. Unlike the BP algorithm, learning in the FC network does not require intensive or repetitive computations. The fact that humans are able to recall patterns seen fleetingly further suggests that neural learning must have an instantaneous training mechanism.

In recent years, the proposition that new directions in signal processing and control will emerge from a study of biological systems has found increasing support [Gazzaniga, 1995]. For example, it has been suggested [Sternberg, 1996] that biological systems perform optimization - as in the protein-folding problem - that is beyond the capability of Turing machines. Likewise, the notion of instantaneous learning provides a new perspective on pattern recognition.

Bibliography

Aleksander, I., ed., 1989. *Neural Computing Architectures: the Design of Brian-like Machines*, MIT Press, Cambridge, Massachusetts.

Aleksander, I., and H. B. Morton, 1990. *An Introduction to Neural Computing*, Chapman & Hall, London.

Aleksander, I., and H. B. Morton, 1991. "A general neural unit: retrievability." *IEE Electronics Letters*, vol. 27, pp. 1776-1778.

Aleksander, I., and H. B. Morton, 1993. *Neurons and Symbols: The stuff that mind is made of*, Chapman & Hall, London.

Aleksander, I., and T. J. Stonham, 1979. "A guide to pattern recognition using random-access memories." *IEEE Journal Computers and Digital Techniques*, vol. 2(1), pp. 29-40.

Aleksander, I., W. V. Thomas, and P. A. Bowden, 1984. "Wisard: a radical step forward in image recognition." *Sensor Review*, July, pp. 120-124.

Anderson, J. A., and E. Rosenfeld, eds., 1988. *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, Massachusetts.

Azoff, E. M., 1994. *Neural Network Time Series Forecasting of Financial Markets*, John Wiley, Chichester.

Baddeley, A., and S. D. Sala, 1996. "Working memory and executive control." *Philosophical Transactions of the Royal Society of London*, vol. 351, pp. 1397-1406.

Becker, S., and Y. LeCun, 1989. "Improving the convergence of back-propagation learning with second-order methods." *Proceedings of the 1988 Connectionist Models Summer School*, pp. 29-37.

Bezdek, J. C., 1981. *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York.

Bezdek, J. C., S. Chuah, and D. Leep, 1986. "Generalized *k*-nearest neighbor rules." *Fuzzy Set Systems*, vol. 18(3), pp. 237-256.

Bezdek, J. C., and S. K. Pal, eds., 1992. *Fuzzy Models for Pattern Recognition: Methods That Search for Structures in Data*, IEEE Press, New York.

Boole, G., 1854. *The laws of thought*, The Open court publishing company, London.

Broomhead, D. S., and D. Lowe, 1988. "Multivariable functional interpolation and adaptive networks." *Complex Systems*, vol. 2, pp. 321-355.

Cacoullos, T., 1966. "Estimation of a multivariate density." *Annals of the Institute of Statistical Mathematics (Tokyo)*, vol. 18, pp. 179-189.

Casdagli, M., 1989. "Nonlinear prediction of chaotic time series." *Physica D*, vol. 35, pp. 335-356.

Casdagli, M. and S. Eubank, eds., 1992. *Nonlinear Modeling and Forecasting*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings vol. XII, Addison-Wesley.

Cho, S. B., and J. H. Kim, 1991. "A fast back-propagation learning method using Aitken's Δ^2 process." *International Journal on Neural Networks*, vol. 2(1), pp. 37-42.

Cover, T. M., 1965. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition." *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 326-334.

Cover, T. M., 1968. "Estimates by nearest neighbor rule." *IEEE Transactions on Information Theory*, vol. IT-14(1), pp. 50-55.

Cover, T. M., and P. E. Hart, 1967. "Nearest neighbor pattern classification." *IEEE Transactions on Information Theory*, vol. IT-13, pp. 21-27.

Daneman, M, and P. M. Merikle, 1996. "Working memory and language comprehension: A meta-analysis." *Psychonomic Bulletin & Review*, vol. 3, pp. 422-432.

Dasarathy, B. V., 1977. "Visiting nearest neighbor rule - A survey of nearest neighbor classification techniques." *Proceedings of the International Conference on Cybernatics*, pp. 630-636.

Dasarathy, B. V., ed., 1991. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, CA.

D'Esposito, M., and M. Grossman, 1996. "The physiological basis of executive function and working memory." *The Neuroscientist*, vol. 2, pp. 345.

Devijver, P. A., 1979. "New error bounds with the nearest neighbor rule." *IEEE Transactions on Information Theory*, vol. IT-25, pp. 749-753.

Devijver, P. A., and Kittler, J., 1982. *Pattern Recognition: A Statistical Approach*, Prentice-Hall, Englewood Cliffs, NJ.

Duda, R. O. and P. E. Hart, 1973. *Pattern Classification and Scene Analysis*, John Wiley, New York.

Fausett, L., 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, NJ.

Fix, E., and Hodges, J., 1951. "Discriminatory analysis: Nonparametric discrimination: Consistency properties." Technical Report 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, TX. Also in [Dasarathy, 1991], pp. 32-39.

Fix, E., and Hodges, J., 1952. "Discriminatory analysis: Small sample performance." Technical Report 21-49-004, USAF School of Aviation Medicine, Randolph Field, TX. Also in [Dasarathy, 1991], pp. 40-56.

Fukunaga, K., 1972. *Introduction to Statistical Pattern Recognition*, Academic Press, Orlando, Florida.

Fuster, J. M., 1995. *Memory in the Cerebral Cortex: An Empirical Approach to Neural Networks in the Human and Nonhuman Primate*, MIT Press, Cambridge, Massachusetts.

Gazzaniga, M. S., ed., 1995. *The Cognitive Neuroscience*, MIT Press, Cambridge, Massachusetts.

Goldman-Rakic, P. S., 1996. "Regional and cellular fractionation of working memory." *Proceedings of the National Academy of Sciences*, vol. 93, pp. 13473.

Hagan, M. T., H. B. Demuth and M. Beale, 1996. *Neural network design*, PWS Publishing Company, Boston.

Hart, P., 1968. "The condensed nearest neighbor rule." *IEEE Transactions on Information Theory*, vol. IT-14, pp. 515-516.

Hastie, T., and R. Tibshirani, 1990. *Generalized Additive Models*, Chapman & Hall, New York.

Haykin, S., 1999. *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ.

Hebb, D., 1949. *The Organization of Behavior*, Wiley, New York.

Hecht-Nelson, R., 1990. *Neurocomputing*, Addison-Wesley, Reading, MA.

Hellman, M. E., 1970. "The nearest neighbor classification rule with a reject option." *IEEE Transactions on Systems, Man, and Cybernatics*, vol. 6(2), pp. 179-185.

Henon, M. A., 1976. "Two-dimensional map with a strange attractor." *Communications in Mathematical Physics*, vol. 50, pp. 69-77.

Hornik, K., M. Stinchcommbe, and H. White, 1989 "Multilayer feedforward networks are universal approximators." *Neural Networks*, vol. 2, pp. 359-368.

Jang, J. R. S., C. T. Sun, and E. Mizutani, 1997. *Neuro-Fuzzy and Soft Computing*, Prentice-Hall, Upper Saddle River, NJ.

Jonides, J. et al., 1996. "Verbal and spatial working memory in humans." *The Psychology of Learning and Motivation*, vol. 35, pp. 43.

Jozwik, A., 1983. "A learning scheme for fuzzy *k*-NN rule." *Pattern Recognition Letters 1*, vol. 1, pp. 287-289.

Kak, S. C., 1992. "New training algorithm in feedforward neural networks." First International Conference on Fuzzy Theory and Technology, Durham, North Carolina, October. Also in Wang, P. P., ed., 1993. *Advance in fuzzy theory and technologies*, Bookwright Press, Durham, North Carolina.

Kak, S. C., 1993. "On training feedforward neural networks." *Pramana J. Physics*, vol. 40, pp. 35-42.

Kak, S. C., 1994. "New algorithms for training feedforward neural networks." *Pattern Recognition Letters*, vol. 15, pp. 295-298.

Kak, S. C., 1995. "On generalization by neural networks." *First International Conference on Computational Intelligence and Neuroscience*, North Carolina.

Kak, S. C., 1998. "On generalization by neural networks." *Information Sciences*, vol. 3, pp. 293-302.

Kak, S. C., 1999. "Faster web search and prediction using instantaneously trained neural networks." *IEEE Intelligent Systems*, vol. 14(5), pp. 2-5.

Kak, S. C. and J. Pastor, 1995. "Neural networks and methods for training neural networks." *U.S. Patent No. 5,426,721*, June 20.

Kartalopoulos, S. V., 1996. *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications*, IEEE Press, New York.

Keller, J. M., M. R. Gray, and J. A. Givens, 1985. "A fuzzy *k*-nearest neighbor algorithm." *IEEE Transactions on Systems, Man, and Cybernatics*, vol. 15(4), pp. 580-585.

Kosko, B., 1992. *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ.

LeCun, Y., 1986. " Learning processes in an asymmetric threshold network." In Bienenstock, E., F. Fogelman-Souli, and G. Weisbuch, eds., 1986. *Discovered Systems and Biological Organization*, Nato ASI Series, F20, Springer-Verlag, Berlin.

Light, W., ed., 1991. *Advances in Numerical Analysis Vol. II: Wavelets, Subdivision Algorithms, and Radial Basis Functions*, Oxford Science Publications, Oxford.

Lin, C. T. and C. S. G. Lee, 1996. *Fuzzy neural systems*, Prentice Hall, Upper Saddle River, NJ.

Looney, C. G., 1997. *Pattern recognition using neural networks*, Oxford University Press, New York.

Mackey, M. C. and L. Glass, 1977. "Oscillation and chaos in physiological control systems." *Science*, vol. 197, pp. 287.

Mason, J. C., and M. G. Cox, eds., 1987. *Algorithms for Approximation*, Clarendon Press, Oxford.

Matlab neural network toolbox user's guide, 1995. The Mathworks, Inc., May.

McClelland, J. L., and D. E. Rumelhart, 1988. *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts.

McCulloch, W. S., and W. Pitts, 1943. "A logical calculus of ideas immanent in nervous activity." *Bull. of Math. Biophys.*, vol. 5, pp. 115-133.

Mead, W. C., R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, L. A. Lee and M. K. O'Rourke, 1992. "Prediction of chaotic time series using CNLS-Net - Example: The Mackey-Glass equation." In [Casdagli and Eubank, 1992], pp. 39-72.

Mehta, A. V., 1997. *Prediction of timing signals in financial series using neural networks*, an M.S. thesis, Louisiana State University, December.

Michalski, R. S., I. Bratko and M. Kubat, eds., 1998. *Machine Learning and Data Mining: Methods and Applications*, John Wiley & Sons, West Sussex.

Minsky, M. L., and S. A. Papert, 1969. *Perceptrons*, MIT Press, Cambridge, Massachusetts.

Moody, J., and C. J. Darken, 1989. "Fast learning in networks of locally-tuned processing units." *Neural computation*, vol. 1, pp. 281-294.

Müller, B., J. Reinhardt and M. T. Strickland, 1995. *Neural networks*, Springer-Verlag, Berlin.

Parker, D. B., 1985. "Learning logic: Casting the cortex of the human brain in silicon." *Technical Report TR-47*, Center for Computational Research in Economics and Management Science, MIT Press, Cambridge, Massachusetts.

Parker, D. B., 1987. "Optimal algorithms for adaptive networks: Second-order backpropagation, second-order direct propagation and second-order Hebbian learning." *Proc. of the IEEE International Conference on Neural Networks*, vol. 2, pp. 593-600.

Parzen, E., 1961. "Mathematical considerations in the estimation of spectra." *Technometrics*, vol. 3, pp. 167-190.

Parzen, E., 1962. "On estimation of a probability density function and mode." *Annals of Mathematical Statistics*, vol. 33, pp. 1065-1076.

Poggio, T., and F. Girosi, 1990. "Networks for approximation and learning." *Proceedings of the IEEE*, vol. 78, pp. 1481-1497.

Powell, M. J. D., 1985. "Radial basis functions for multivariable interpolation: a review." *IMA Conference on Algorithms for the Approximation of Functions and Data*, RMCS, Shrivenham, England. Also in [Mason and Cox, 1987], pp. 143-167.

Powell, M. J. D., 1988. "Radial basis function approximations to Polynomials." *Numerical Analysis 1987 Proceedings*, pp. 223-241, Dundee, UK.

Powell, M. J. D., 1992. "The theory of radial basis function approximation in 1990." In [Light, 1991], pp. 105-210.

Rosenblatt, F., 1958. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, pp. 386-408. Reprinted in [Anderson and Rosenfeld, 1988], pp. 92-114.

Rosenblatt, F., 1959. "Two theorems of statistical separability in the perceptron." *Mechanization of Thought Processes: Proceeding of a Symposium Held at the National Physical Laboratory, November 1958*, London.

Rosenblatt, F., 1960a. "Perceptron simulation experiments." *Proceedings of the Institute of Radio Engineers*, vol. 48, pp. 301-309.

Rosenblatt, F., 1960b. "On the convergence of reinforcement procedures in simple perceptron." Cornell Aeronautical Laboratory Report, VG-1196-G-4, Buffalo, NY.

Rosenblatt, F., 1962. *Principles of Neurodynamics*, Spartan Books, Washington, DC.

Rosenblatt, M., 1956. "Remarks on some non-parametric estimates of a density function." *Annals of Mathematical Statistics*, vol. 27, pp. 832-837.

Rosenblatt, M., 1970. "Density estimates and Markov sequence." In M. Puri, ed., *Nonparametric Techniques in Statistical Inference*, Cambridge University Press, London, pp. 199-213.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986a. "Learning internal representations by error propagation." In [Rumelhart and McClelland, 1986], pp. 318-362. Reprinted in [Anderson and Rosenfeld, 1988], pp. 675-695.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986b. "Learning representations of back-propagation errors." *Nature (London)*, vol. 323, pp. 533-536. Reprinted in [Anderson and Rosenfeld, 1988], pp. 696-699.

Rumelhart, D. E., and J. L. McClelland, eds., 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, MIT Press, Cambridge, Massachusetts.

Ryan, T. P., 1997. *Modern Regression Methods*, John Wiley, New York.

Shu, B. and S. C. Kak, 1999. "A neural network-based intelligent metasearch engine." To appear in *Information Science*.

Souza, G. A. and S. C. Kak, 1998. "Dynamic radius allocation in corner classification neural network." *Third International Conference on Computational Intelligence and Neuroscience*, North Carolina.

Specht, D. F., 1988. "Probabilistic neural networks for classification, mapping, or associative memory." *Proceedings of the IEEE International Conference on Neural Networks*, vol. 1, pp. 525-532.

Specht, D. F., 1990. "Probabilistic neural networks." *Neural Networks*, vol. 3(1), pp. 109-118.

Specht, D. F., 1991. "A general regression neural network." *IEEE Transactions on Neural Networks*, vol 2, pp. 568-576.

Specht, D. F., 1996. "Probabilistic neural networks and general regression neural networks." In Chen, C. H., ed., 1996. *Fuzzy Logic and Neural Network Handbook*, McGraw-Hill, New York, Chapter 3.

Sternberg, M. J. E., ed., 1996. *Protein Structure Prediction*, Oxford University Press, Oxford.

Stokbro, K. and D. K. Umberger, 1992. "Forecasting with weighted maps." In [Casdagli and Eubank, 1992], pp. 73-94.

Tang, K. W., 1997. *CC4: A new corner classification approach to neural network training*, an M.S. thesis, Louisiana State University.

Tang, K. W. and S. C. Kak, 1997. "Corner classification that allows inhibitory output weights." *Second International Conference on Computational Intelligence and Neuroscience*, North Carolina.

Tang, K. W. and S. C. Kak, 1998. "A new corner classification approach to neural network training." *Circuits Systems Signal Processing*, vol. 17, No. 4, pp. 459-469.

Thompson, J. M. T., and H. B. Stewart, 1986. *Nonlinear Dynamics and Chaos*, John Wiley, Chichester.

Tomek, I., 1976. "A generalization of the *k*-NN rule." *IEEE Transactions on Systems, Man, and Cybernatics*, vol. 6(2), pp. 121-126.

Valiant, L. G., 1994. *Circuits of the Mind*. Oxford University Press, New York.

Wand, M. P., and M. C. Jones, 1995. *Kernel Smoothing*, Chapman & Hall, London.

Wasserman, P. D., 1989. *Neural computing*, Van Nostrand Reinhold, New York.

Watanabe, S., 1985. *Pattern Recognition: Human and Mechanical*, John Wiley, New York.

Whittle, P., 1958. "On the smoothing of probability density functions." *Journal of the Royal Statistical Society, Series B*, vol. 20, pp. 334-343.

Wickelgren, I., 1997. "Getting a grasp on working memory." *Science*, vol. 275, pp. 1580-1582.

Xu, L., A. Krzyzak, and A. Yuille, 1994. "On radial basis function nets and kernel regression: Statistical consistency, convergency rates, and receptive field size." *Neural Networks*, vol. 7, pp. 609-628.

Zadeh, L. A., 1965. "Fuzzy sets." *Information and Control*, vol. 8, pp. 338-353.

Zurada, J. M., 1992. *Artificial Neural Networks*, West Publishing, St. Paul, MN.

Vita

Kun Won Tang received his bachelor of engineering degree in Electrical Engineering from the University of Malaya, Kuala Lumpur, Malaysia, in 1978. After graduation, he spent many years in the business of designing and implementing microprocessor- and PC-based systems in Malaysia. In August 1996, he joined the graduate program in the Department of Electrical and Computer Engineering, Louisiana State University and received his master of science degree in May 1997. He is currently a candidate for the degree of Doctor of Philosophy which will be conferred in December 1999.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Kun Won Tang

Major Field: Electrical Engineering

Title of Dissertation: Instantaneous Learning Neural Networks

Approved:

Subhash Kak
Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

John M. Tyler

J. Ramsey

D. L. JC

Fred D. Denny

Robert Wilkinson

Date of Examination:

October 19, 1999